Software Diagnostics Developer Edition 1.0

© 2010 Software Diagnostics

SD Developer Edition

User Guide for Software Diagnostics Developer Edition 1.0

by Software Diagnostics

This document describes the features of SD DevEdition and provides quickstart tutorials to enable you to apply the tool right away.

Throughout the manual, the abbreviation SD DevEdition is used to refer to Software Diagnostics Developer Edition.

Thank you for using Software Diagnostics Developer Edition.

SD Developer Edition

© 2010 Software Diagnostics

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Author and Company Informations:

Software Diagnostics

August Bebel Straße 26-53
D-14482 Potsdam
Germany

CEO: Marc Hildebrandt
Johannes Bohnet

Amtsgericht Potsdam
HRB 22123 P
Finanzamt Potsdam Stadt

Phone: +49 331 7062340
Fax: +49 331 7062341

EMail: mail@softwarediagnostics.com
Internet: http://www.softwarediagnostics.com

Printed: Mai 2010 in Potsdam, Germany
Version: SD Developer Edition 1.0.5400

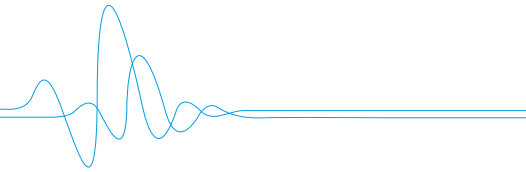


Table of Contents

Part I Quickstart 7
1.1 Tracing Methods 7
1.2 Tracing Strategies 8

Part II Operational Requirements and Installation 10
2.1 Operational Requirements 10
2.2 Installation 10

Part III Prepare Your Software System 12
3.1 Initial Preparation 12

Part IV Trace Your System's Behavior 15
4.1 Optional Preparation Step 15
4.2 Start Tracing 15
4.2.1 TraceOver 15
4.2.2 AttachToProcess 18
4.2.3 StartWithTracing 18
4.3 SD Recorder 19
4.4 Polluting Functions 20
4.5 Probes 21
4.5.1 DataProbes 21
4.5.2 MemAccessProbes (experimental) 24
4.5.3 ParamProbes 25

Part V Import Traces 28
5.1 Create Project Dialog 28
5.2 Import Dialog 28
5.3 Manage and Delete Traces 29

Part VI Visual Analysis 32
6.1 CallStackView 32
6.2 CallGraphView 34
6.3 SingleCallView 34
6.4 SearchView 35

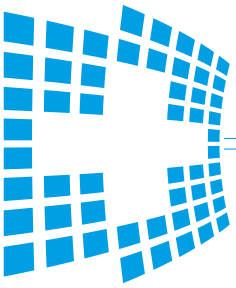
Part VII Tutorials 37
7.1 Bug: Erroneous Database Entry 37
7.2 Bug: XML Content Written in Wrong Order 39
7.3 Feature Location: Privacy Data in Google Chrome 43

© 2010 Software Diagnostics

Part VIII Experimental Features 47

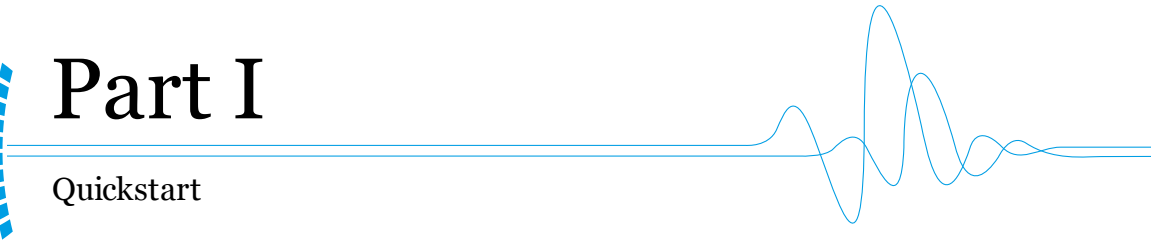
Part IX Problems and Known Issues 49

© 2010 Software Diagnostics



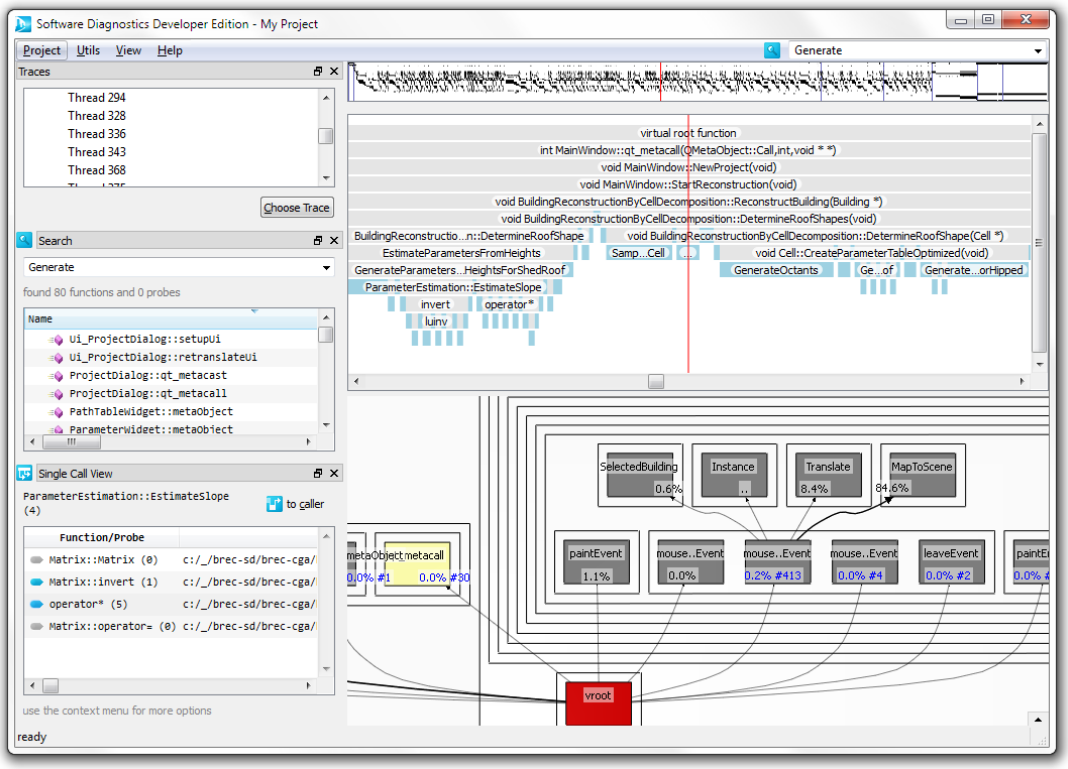
Part I

Quickstart



1 Quickstart

This chapter briefly describes the most important steps for using SD Dev Edition.



1.1 Tracing Methods

There are three different ways to trace your application. All three methods are started by clicking the Tracing icon in the Visual Studio toolbar. The methods available are dependent on the state of the Visual Studio debugger.

(1) StartWithTracing
You haven't started your app yet. Click the Tracing icon. Click Start Debugger. The app starts in tracing mode. Record a trace with the SD Recorder.

You always should start tracing with a low polluting functions threshold, e.g., 10.

(2) AttachToProcess
Your app is running, however, not under control of the Visual Studio debugger. For instance, you have started it with Visual Studio's Ctrl-F5 StartWithoutDebugging. Click Tracing icon. Choose the process ID of your app. Click AttachToProcess. With the SD Recorder you record a trace.

You always should start tracing with a small polluting functions threshold, e.g., 10.

(3) TraceOver
Your app is currently debugged by the Visual Studio debugger and it is in break mode. You

© 2010 Software Diagnostics

are probably performing StepInto and StepOver operations to navigate through the execution. If you are at a point in the execution where you are not sure if StepInto is a waste of time or if StepOver skips an important part of the execution, then use TraceOver. Click the Tracing icon. Choose the point in the code where tracing should be stopped by clicking the more and less buttons. Choose one of the TraceOver options.

Be sure that the control flow passes the code line where the blue region ends. If the control flow misses this point in the code, TraceOver won't stop. You may be concerned when TraceOver takes a long time and nothing happens. In this case, use the Stop icon in the Visual Studio toolbar to interrupt tracing.

- Prepare Your Software System -

After preparing your software system, you should at least exclude compiler-generated functions and functions of the Standard Template Library STL from being traced. Otherwise, you get huge traces with lots of information on compiler and STL internals. For details read the respective chapter.

1.2 Tracing Strategies

The main strategy is to identify the core part of execution that is of interest for your current task. Hence, the goal is to do a TraceOver of a reasonably small number of function calls, e.g., 10.000, and trace them with full detail, e.g., with detailed ParamProbes.

- Find a good place for a breakpoint -

Start your app without Visual Studio debugging support (Ctrl-F5). Do the necessary interactions with your app. Do AttachToProcess. Choose a low polluting functions threshold, e.g., 10. Click Record in SD Recorder. Interact with your app. Click Stop. Analyze the trace. Identify the interesting part of execution, i.e., the responsible call that triggers this part of the execution. Switch to Visual Studio's code view via the context menu of the call and place a breakpoint.

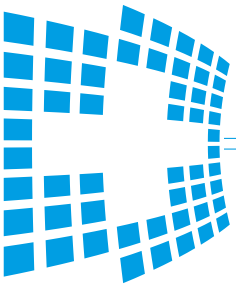
Click the (Software Diagnostics) Stop icon in the Visual Studio toolbar and switch the process to the Visual Studio debugger. Click run in Visual Studio and interact with your app such that the control flow hits the breakpoint.

Do TraceOver. If you don't know how many calls are triggered by TraceOver try a polluting functions threshold of 50-100 and do not use ParamProbes. (With ParamProbes the polluting functions mechanism won't find many of these functions.) Analyze the trace and find an even more detailed part of the execution that is of interest.

Do TraceOver again. If you are sure that you trace less than 10.000 calls, use DetailedParamProbes to record not only the calls but also all parameter values that are passed into the functions. Analyze the trace.

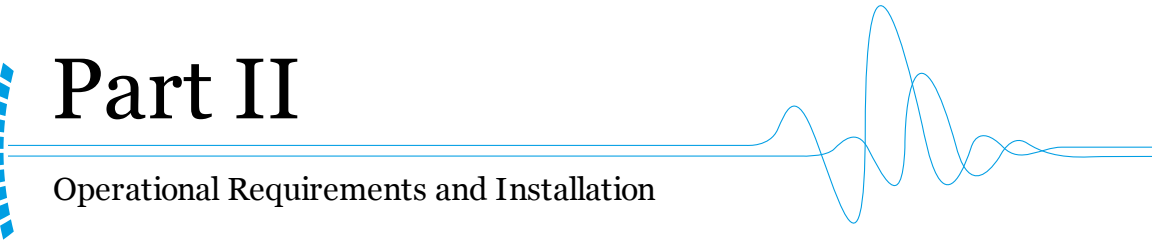
The search view also searches ParamProbes which lets you search for suspicious parameter values.

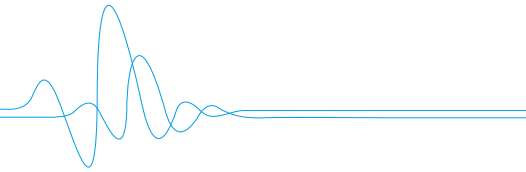
© 2010 Software Diagnostics



Part II

Operational Requirements and Installation





2 Operational Requirements and Installation

This chapter outlines the operational requirements of SD Dev Edition and guides you through the installation process.

2.1 Operational Requirements

SD Dev Edition depends on software packages of other vendors. You need to have installed at least:

- Microsoft Windows XP, Microsoft Windows Vista or Microsoft Windows 7
- Microsoft Visual Studio 2008
- x86 compatible environment (on x64 it runs as an x86 application)
- .NET Framework 3.5 (latest updates)
- Microsoft Debugging Tools for Windows x86 (tested with 6.11.1.404)

SD Dev Edition requires minimal hardware support:

- Graphics Card with OpenGL 1.4 support

2.2 Installation

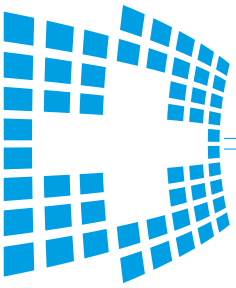
If you have a previous version of SD Dev Edition, please uninstall this version first before installing the new version. The uninstall executable is found in the start menu.

To install SD Dev Edition, run the installer executable. The following components are installed:

- SD Dev Edition: the main application is installed to "Program Files/Software Diagnostics/Developer Edition" ("Program Files (x86)/Software Diagnostics/Developer Edition" on x64 operating systems) [environment variable SD_HOME]
- sd_vsaddin_2008.dll: is the Microsoft Visual Studio addin component of SD Dev Edition and is placed in "My Documents/Visual Studio 2008/Addins/".
- SD Recorder: is the trace recording component. It is installed into "My Documents/Traces/" [environment variable SD_TRACING].
- All traces and other documents while using the SD Dev Edition are stored in the location: "My Documents/Traces/" [environment variable SD_TRACING].
- The installer creates the environment variables SD_HOME and SD_TRACING.

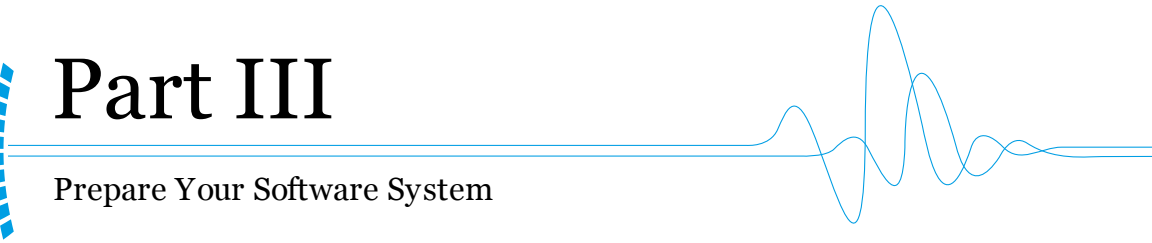
If you encounter problems, look at the Problems and Known Issues chapter.

© 2010 Software Diagnostics



Part III

Prepare Your Software System



3 Prepare Your Software System

Before you can use SD Dev Edition to analyze the dynamics of your software system, you need to perform an initial preparation step.

3.1 Initial Preparation

SD Dev Edition uses dynamic analysis techniques to create a trace of your system's internal behavior. To enable your system, you need to perform an initial automated preparation step. This step induces a complete rebuild of your code. However, you only need to do this once. After this, you can develop your code as usual - there is no increased build time or performance overhead at runtime.

1. Open your solution (your code) in Microsoft Visual Studio.

2. Click the prepare solution icon in the Microsoft Visual Studio SDAddin Toolbar (stethoscope symbol).

Microsoft Visual Studio SDAddin Toolbar

3. Choose the configuration that you work with (e.g., Debug) and check all projects. You have the option to repeat this for other configurations.

[EXPERIMENTAL]

Tracing optimized code, which is typically done in the Release configuration, is an experimental feature at the moment.

4. Click "Patch projects".

5. Perform a full build of your code.

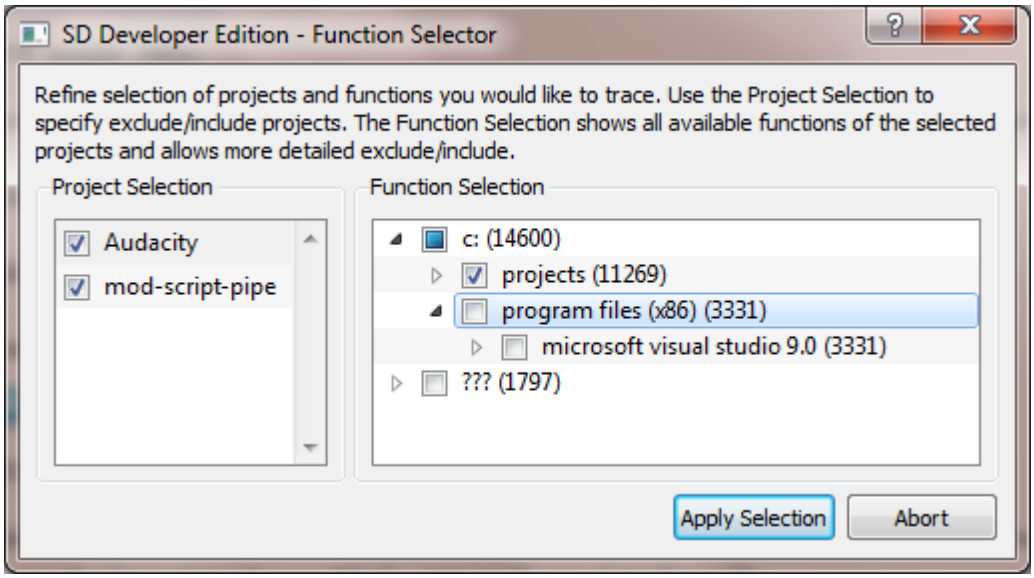
6. Click the function selection icon in the Microsoft Visual Studio SDAddin Toolbar (f symbol).

Microsoft Visual Studio SDAddin Toolbar

7. Check all projects (exe/dll) and uncheck all directories that you do not want to trace. You typically do not want to trace compiler generated functions ("???") and functions from

© 2010 Software Diagnostics

the standard template library STL ("…/microsoft visual studio 9.0").

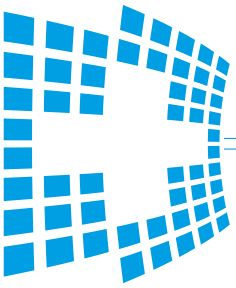

SD Developer Edition - Function Selector
Refine selection of projects and functions you would like to trace. Use the Project Selection to specify exclude/include projects. The Function Selection shows all available functions of the selected projects and allows more detailed exclude/include.
Project Selection
Audacity
mod-script-pipe
Function Selection
c: (14600)
projects (11269)
program files (x86) (3331)
microsoft visual studio 9.0 (3331)
??? (1797)
Apply Selection
Abort

FunctionSelector Dialog

8. Click "Apply Selection".

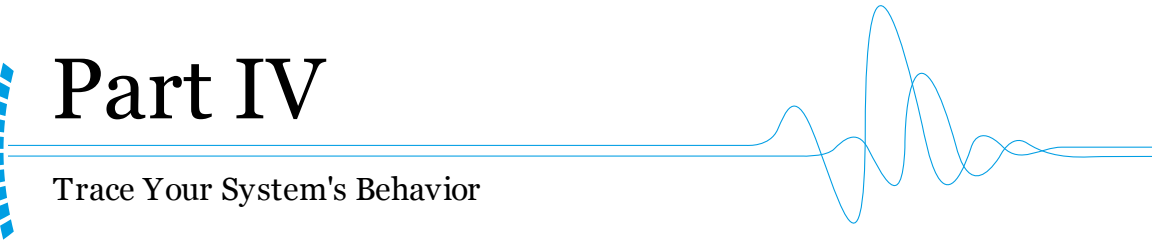
As you see in the FunctionSelector Dialog, you can select parts of 3rd-party libraries. However, note that the 3rd-party libraries only appear in the dialog because their header files contain compilable code, e.g., due to inlined code or to template usage. You will typically deactivate tracing for these functions because they implement low-level, utility functionality. For instance, the STL internals as part of the "microsoft visual studio 9.0" folder in the figure above typically won't be of interest. Unfortunately, "interesting" API calls to 3rd-party libraries are not traceable if you don't have the source code and you don't integrate their Microsoft Visual Studio projects into your application's Microsoft Visual Studio solution. A workaround could be to encapsulate a specific API function of interest with a wrapper function that is part of your application. This way, the call to the wrapper function is traced which provide equivalent information than directly tracing the call to the original API function.

© 2010 Software Diagnostics



Part IV

Trace Your System's Behavior



4 Trace Your System's Behavior

With SD DevEdition you are able to "see" the inner workings of your software system at runtime. The behavior is first captured as a trace and then visualized.

This chapter explains how to create a trace.

4.1 Optional Preparation Step

As an optional step, you may want to restrict the functions whose calls are captured in a trace. To do so, follow the instructions given in the preparation chapter.

4.2 Start Tracing

Click the StartTracing icon in the Microsoft Visual Studio SDAddin Toolbar. Depending on the state of Microsoft Visual Studio, you have 3 different tracing options:

1. TraceOver

2. AttachToProcess

3. StartWithTracing

Be aware that tracing refers to the current "StartUp Project" in your Visual Studio solution. Hence, assure that the StartUp Project is set correctly before applying tracing.

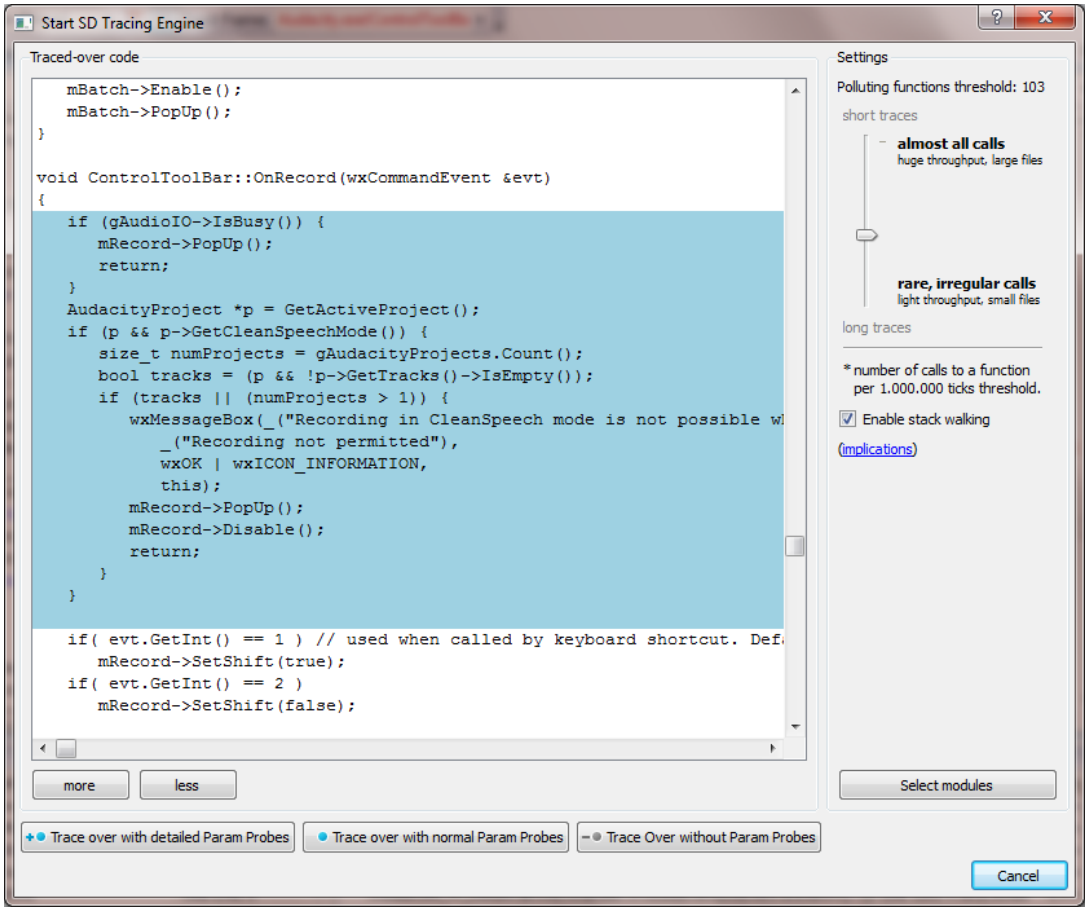
4.2.1 TraceOver

To use TraceOver, you need to run your application with the Microsoft Visual Studio Debugger. Just debug as usual and stop your application at a breakpoint. You have the option of performing the usual Microsoft Visual Studio step-over or step-into operations to move forward within the execution.

Debugging typically means that you have to decide whether to follow a call (step-into) or to skip it (step-over). At some point, this decision is difficult: Step-into would probably involve wasting time because you dive into too detailed an execution, whereas step-over probably moves you over important parts of the execution.

In these situations, use TraceOver. Click for the TraceOver dialog.

© 2010 Software Diagnostics



Setting the Code Range

Now define the code range to trace. The last line of the selection defines a "stop tracing breakpoint". Hence, it defines when tracing is finished.

Be sure that control passes the last code line of the selected range.

If this is not the case, TraceOver will not stop automatically. You then have to stop it manually by clicking the stop icon in the Microsoft Visual Studio SDAddin Toolbar.

Polluting Function Threshold

The polluting function mechanism permits tracing with different tracing granularities. For a detailed description see the polluting functions section.

© 2010 Software Diagnostics

Param Probes

Clicking "TraceOver without ParamProbes" traces function calls. Additionally, DataProbes and MemAccessProbes are traced if they have been set. After tracing, the trace is automatically imported into SD Dev Edition and visualized.

If you want to trace the parameter values that are passed into the functions, click either "TraceOver with normal ParamProbes" or "TraceOver with detailed ParamProbes". Both tracing types slow down execution significantly. Hence, apply them only if you are sure that less than approx. 50.000 function calls are executed in the selected code range. Be aware that due to the slow down, the polluting function mechanisms will most likely not identify a single function as polluting. Hence, you trace every call which in turn might increase the number of traced calls.

"TraceOver with normal ParamProbes": For all function calls, the parameter values are traced. For simple data types such as int, char*, float, etc. this tracing technique is the appropriate choice. Passed objects, however, are not resolved. Only the pointer value is shown.

"TraceOver with detailed ParamProbes": In contrast to normal ParamProbes, the nested structure of a parameter is resolved. Object pointers, e.g., a "this" pointer, are not just pointer values, but all their attribute values are resolved. This technique is significantly slower than normal ParamProbes.

Select Modules

You may choose which loaded dlls to trace. Additionally, you may exclude functions.

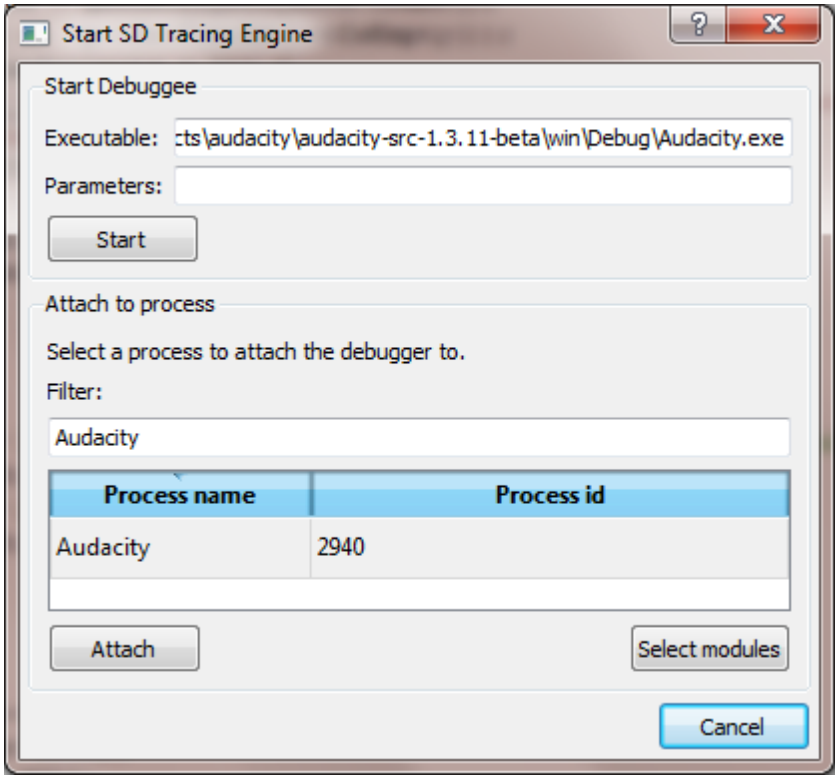
Visual Studio Debugging

After a TraceOver operation, the Visual Studio's stop button does not kill the app anymore. In contrast, Visual Studio detaches itself and keeps the app running.

© 2010 Software Diagnostics

4.2.2 AttachToProcess

AttachToProcess is available via clicking on in the Microsoft Visual Studio SDAddin Toolbar if you started your application as a freely running process, i.e., without being under the control of the Microsoft Visual Studiodebugger. If you started your app with the Microsoft Visual Studiodebugger, simply detach the Microsoft Visual Studiodebugger.



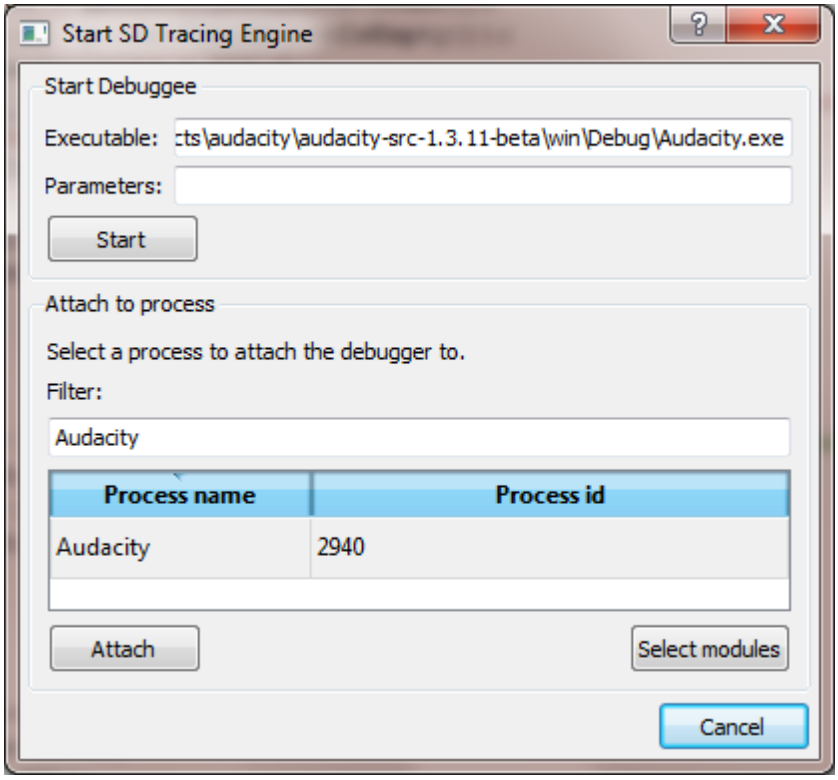
Choose the process ID that corresponds to your application. You can use the filter text field to quickly find the process. Optionally, choose "select modules" to define the loaded dlls and the functions to trace.

After attaching the tracing engine to the process, the SD Recorder pops up. It is used to start and stop the tracing.

4.2.3 StartWithTracing

StartWithTracing is available via clicking on in the Microsoft Visual Studio SDAddin Toolbar if your application is not started yet.

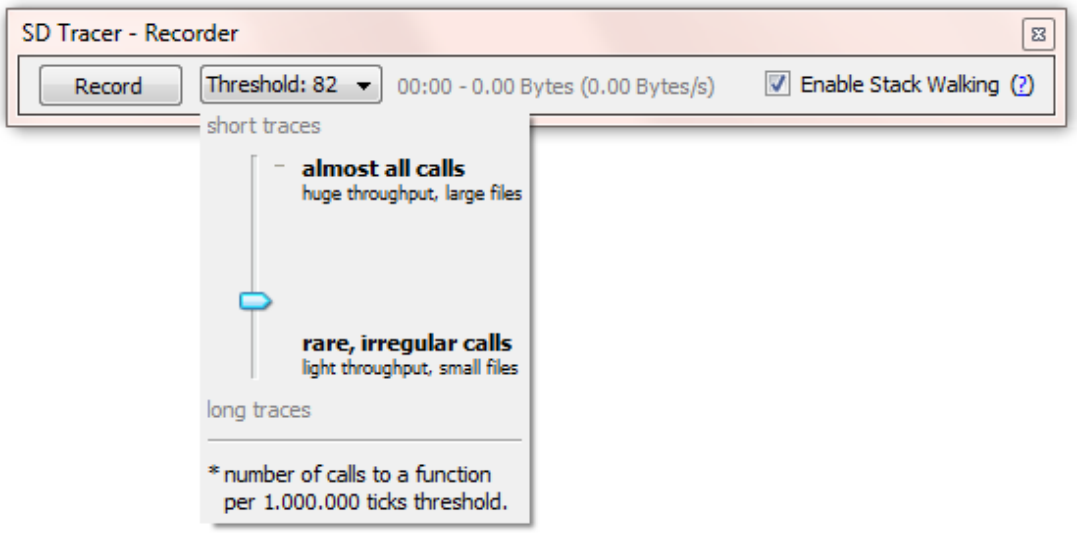
© 2010 Software Diagnostics



Start your application, optionally with command line parameters. After this, the application is started and the SD Recorder pops up, which is used to start and stop the tracing.

4.3 SD Recorder

The SD Recorder is used to start and stop tracing similar to a screen capturing tool.



You have the option of defining a threshold value for the polluting functions mechanism.

© 2010 Software Diagnostics

Click record and then stop to create a trace. You can create several traces without needing to restart your application.

After you have stopped tracing, the trace is automatically imported into SD Dev Edition and visualized.

4.4 Polluting Functions

The polluting functions mechanism is a technique to cope with the vast amount of runtime data that is accumulated during long tracing times. The mechanism enables you to trace for large time ranges (up to 30 minutes).

The key idea is to apply a data mining technique during tracing that identifies polluting functions and to automatically disable the tracing technique for them. The data mining technique is steered by a threshold value: the polluting functions threshold.

You should adjust the threshold value according to the tracing time that you expect. A lower threshold value means that fewer functions are contained in the trace. However, the ones that give you a broad picture of the behavior are guaranteed to remain in the trace. There is a trade-off: Trace for a long time period but receive a coarse-grained picture of the behavior only, or trace for a short time period and get full details.

Examples:

- Expected tracing time: 10 minutes
  Threshold value: 2
- Expected tracing time: 1 minute
  Threshold value: 20
- Expected tracing time: 0.1 seconds
  Threshold value: max

The data mining technique is to some degree sensitive to the algorithms that are implemented in your code. Hence, you need to choose the threshold value that is appropriate for your code and your problem at hand.

A good strategy is to first use SD Recorder and trace with a low threshold value (e.g., 10) to get an overview of the execution, then analyze the trace and identify a key call that triggers the functionality that is of interest. Second, set a breakpoint to the corresponding function and use Microsoft Visual Studio Debugger to navigate to this point in the execution. Third, apply TraceOver to obtain a highly detailed trace that only covers the part of the execution that is of interest.

© 2010 Software Diagnostics

4.5 Probes

Probes are tracing components that enrich control flow information (sequence of function calls) by information on the system state. 3 types of probes are available:

- DataProbes
- MemAccessProbes
- ParamProbes

4.5.1 DataProbes

A DataProbe is attached to a code line and a variable that is reachable from this line. The DataProbe traces the value of the variable when the control flow passes the code line.

Hence, DataProbes are an automated way of the usually tedious debugging cycle of setting a breakpoint, breaking execution at a specific code line, observing the variable's value, and continuing execution until the breakpoint is hit again.

Another common technique to collect the data that DataProbes provide, is to integrate cout() or printf() statements into the code. With DataProbes, you don't need to change the code, and in particular, you don't need to recompile it. Just add a DataProbe where a variable is of interest.

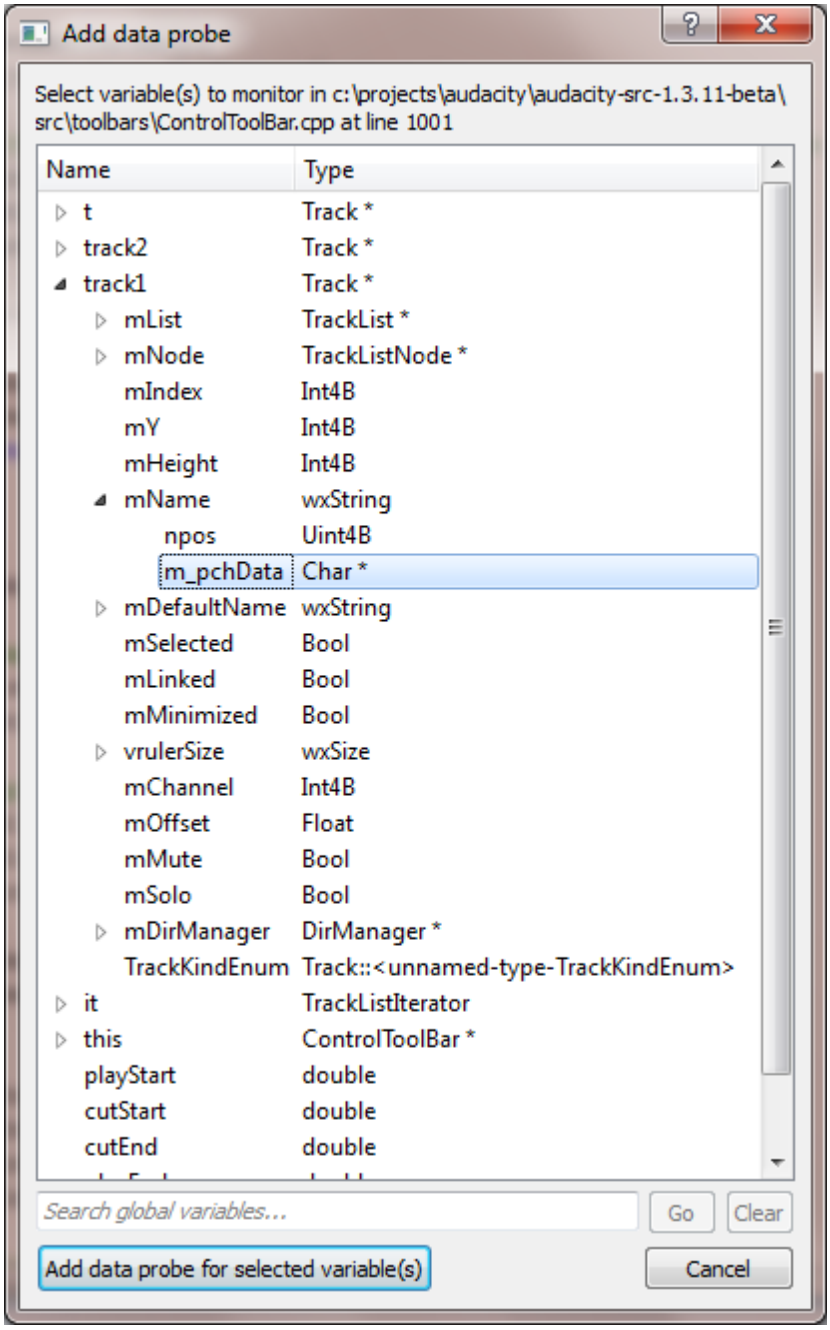
Setting a DataProbe

The prerequisite to setting a probe is that your application is not running.

In the Microsoft Visual Studio code view put the cursor on the code line where you want to trace a variable. Right click and choose: "Add DataProbe".

You can only set DataProbes on code lines that are part of the executable that is currently set as StartUp Project your Microsoft Visual Studio solution.

© 2010 Software Diagnostics



Local Variables

You can recursively crawl through all variables that are accessible at the code line. Choose a top-level or nested variable.

Global Variables (experimental)

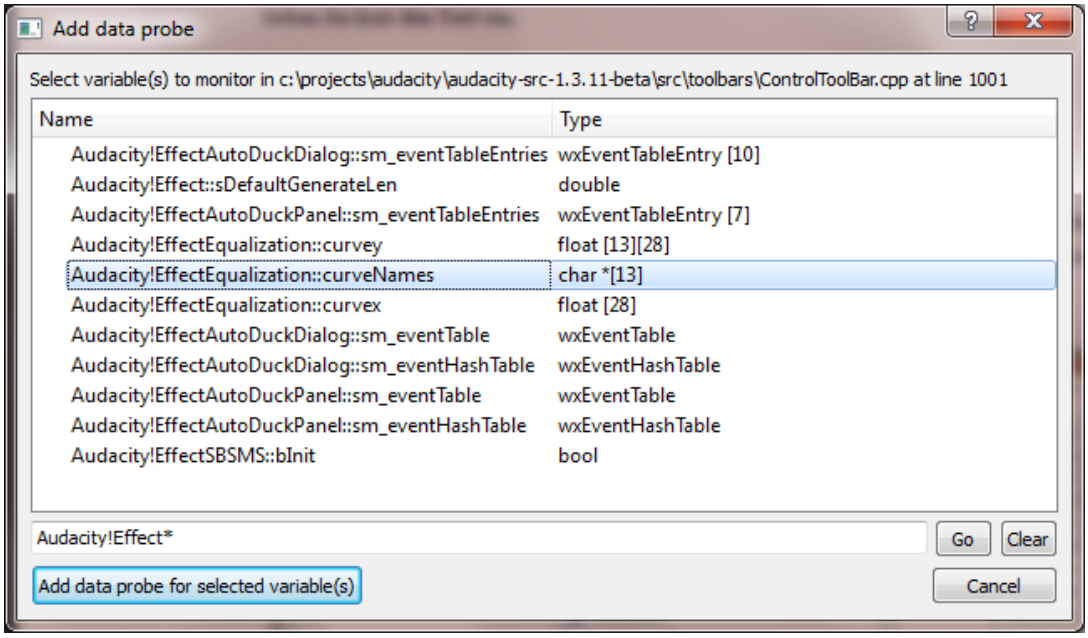
Alternatively, type a search pattern to search for a global variable.

Don't start with * if you have a large code base. Retrieving all matching global

© 2010 Software Diagnostics

variables may take a considerable amount of time. It's better to start with choosing a more restricted pattern.

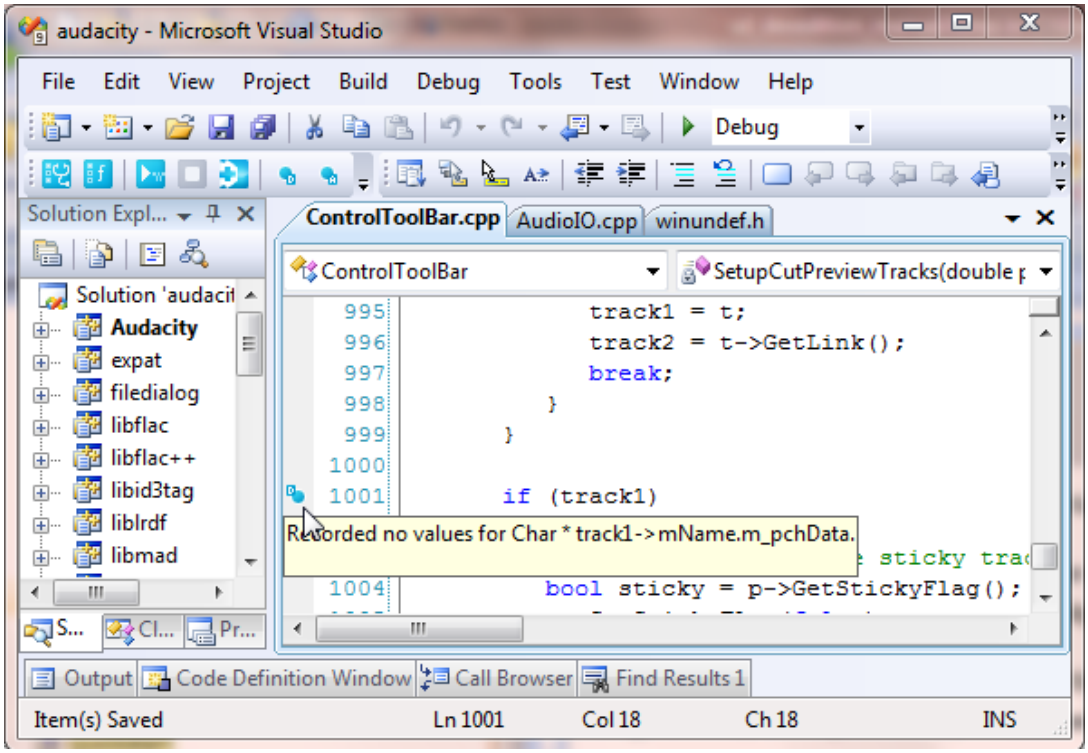
To cancel the operation if it takes too long, click on the SD tray icon in Window's menu bar (located near Window's clock icon).



Deleting a DataProbe

After setting a DataProbe, a marker is shown at the corresponding line in the Microsoft Visual Studio code view. To delete the DataProbe simply click on the marker.

© 2010 Software Diagnostics



4.5.2 MemAccessProbes (experimental)

MemAccessProbes are used to observe read and write operations on memory locations. They are in particular helpful to answer questions in situations like this:

- Why does variable X contain an erroneous value?
- I know that the value was still OK when it was accessed at a previous execution point via variable Y!
- When and why did the erroneous value come in?

To use MemAccessProbes you define:

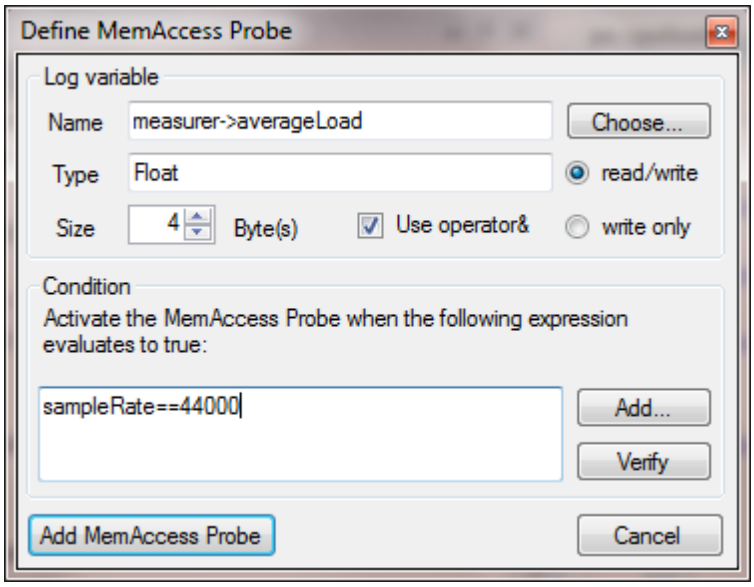
- the code line where the MemAccessProbe is activated,
- the condition when the MemAccessProbe is activated, and
- the variable whose memory location is observed.

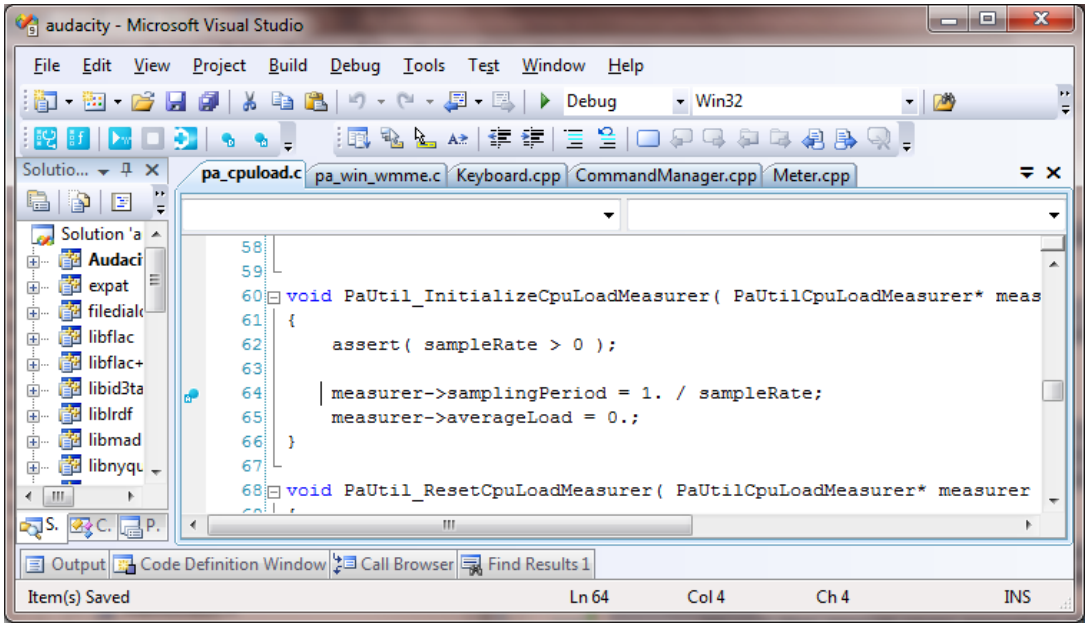
In the screenshots, the MemAccessProbe gets activated when control passes code line 64 of file

© 2010 Software Diagnostics

pa_cpuload.c and the variable sampleRate is equal to 44000.

After this, every access on the memory location is recorded where the value of the variable measurer->averageLoad is stored.





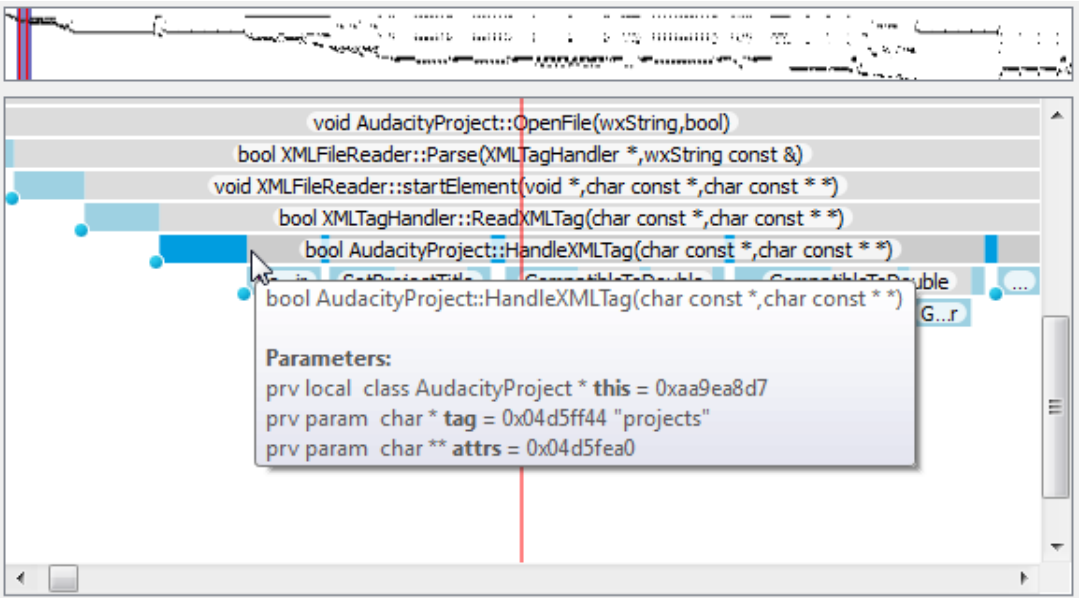
4.5.3 ParamProbes

ParamProbes record the parameter values that are passed into the function for each function call. ParamProbes are not attached to individual code lines such as DataProbes or MemAccessProbes. ParamProbes are activated globally in the TraceOver dialog. Two types of ParamProbes exist:

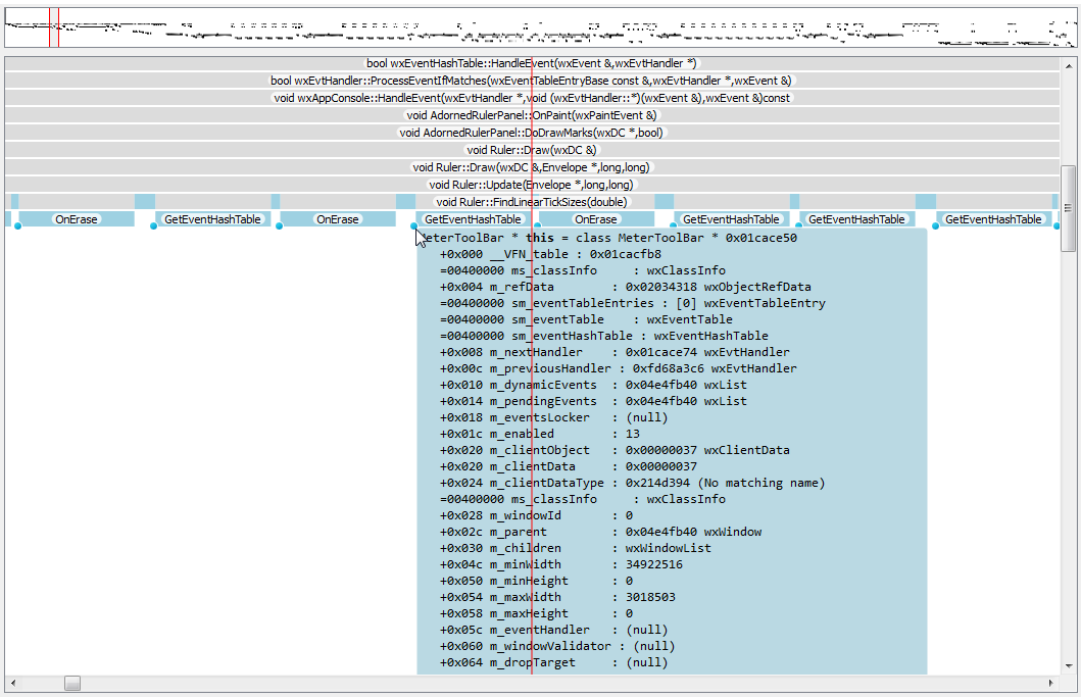
- ParamProbes: This is the light-weight approach. For each function call, the plain

© 2010 Software Diagnostics

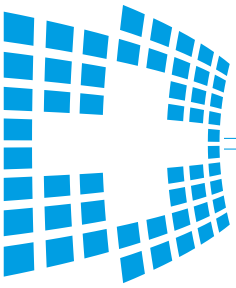
parameter values are recorded. If an object pointer is passed into a function, for instance, only the memory address is recorded. For primitive data types such as int, float, char*, the "real" values are recorded.


void AudacityProject::OpenFile(wxString,bool)
bool XMLFileReader::Parse(XMLTagHandler *,wxString const &)
void XMLFileReader::startElement(void *,char const *,char const * *)
bool XMLTagHandler::ReadXMLTag(char const *,char const * *)
bool AudacityProject::HandleXMLTag(char const *,char const * *)
bool AudacityProject::HandleXMLTag(char const *,char const * *)
Parameters:
prv local class AudacityProject * this = 0xaa9ea8d7
prv param char * tag = 0x04d5ff44 "projects"
prv param char ** attrs = 0x04d5fea0

- DetailedParamProbes: This is the heavy-weight approach: For each function call, the parameter value and all further referenced variable values are recorded. If an object pointer is passed into a function, for instance, the memory address and all attributes of the object are recorded. This is especially useful as one can look into this pointer. Recording overhead for DetailedParamProbes is approximately 10 times larger than for normal ParamProbes.

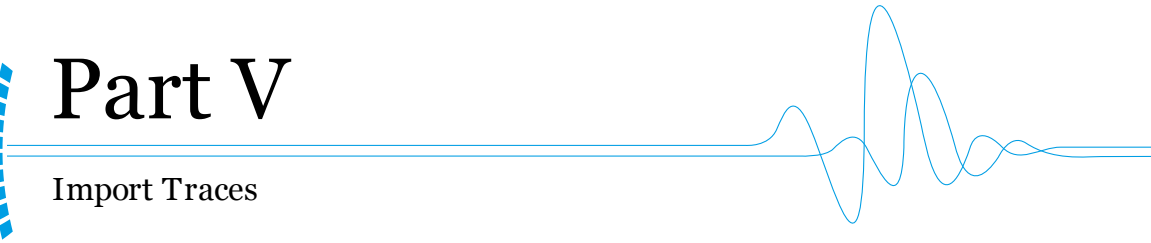


© 2010 Software Diagnostics



Part V

Import Traces



5 Import Traces

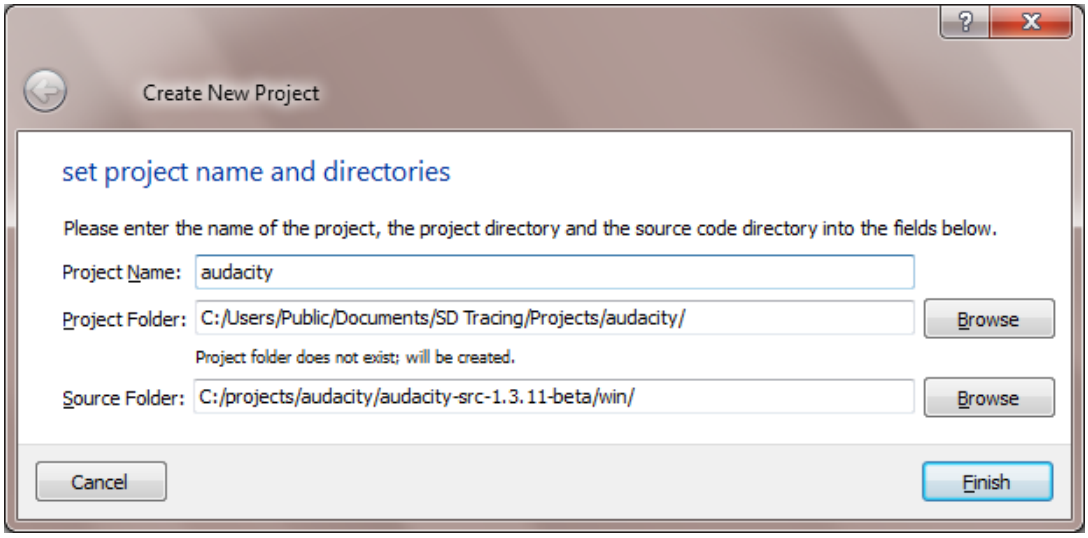
After tracing your application (see Trace Your System's Behavior), SD Dev Edition starts automatically. If this is the first trace, you need to create an SD project for your current Visual Studio solution.

For each recorded trace, the Import Dialog of SD Dev Edition automatically pops up, which imports the trace into the SD Dev Edition database and prepares it for visual analysis.

If there are many DLLs used by your application, it might take a while until the debug information is loaded and the trace import begins. The next trace import will be much faster, however, when the corresponding pdb files are cached by the operating system's disk caching mechanism.

5.1 Create Project Dialog

When importing the very first trace for your current Visual Studio solution, the Create Project Dialog pops up. Typically, you should keep the default values for "Name" and "Project Folder". The "Source Folder" might be shortened to the root folder of your source code. In the case shown below, one would remove the "win", for instance.



5.2 Import Dialog

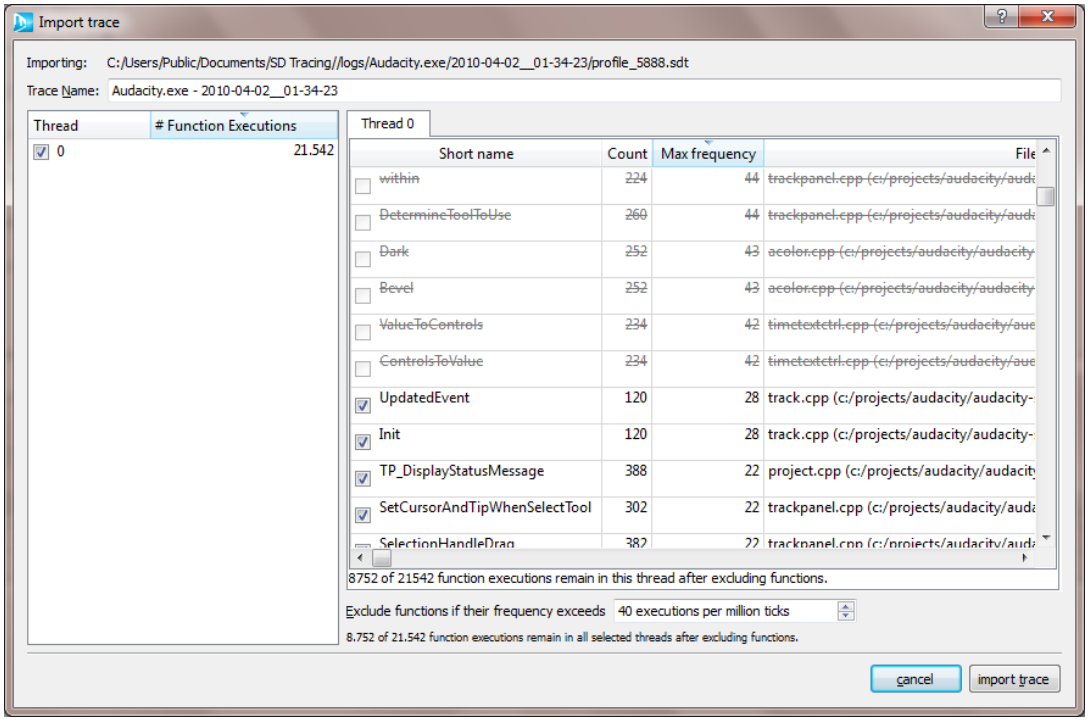
After recording a trace, the Import Dialog pops up. The dialog is used to choose the threads to import and to reduce the size of the trace. SD Dev Edition currently supports traces with a maximum size of 200.000 calls. Hence, you have to eliminate the functions with an extraordinary high call count or polluting function value in the Import Dialog .

© 2010 Software Diagnostics

First, choose the threads to import. Second, choose for every thread which functions to eliminate from the trace until each thread has a reasonably small number of calls.

There are three ways to eliminate functions:

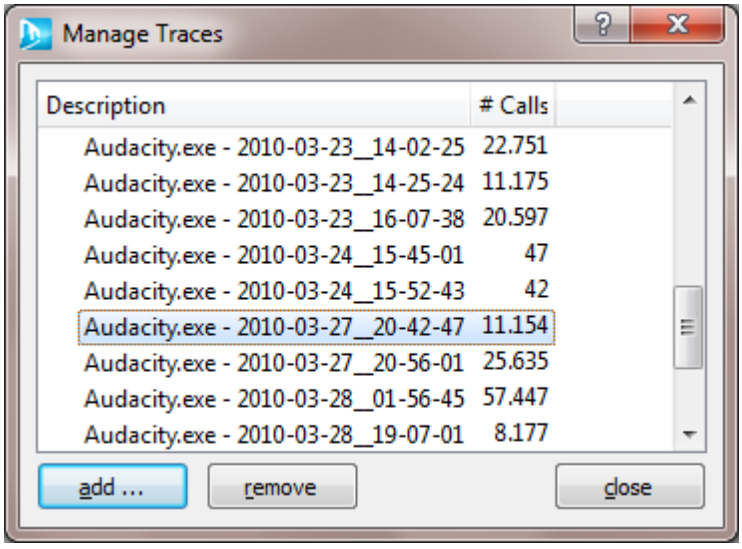
- Eliminate functions with polluting function values above a certain threshold. For this, adjust the threshold value.
- Eliminate functions with high call counts. For this, sort the list, mark a function, hold the shift key, mark another function, and deselect the functions by pressing the "space" key.
- Deselect individual functions.


Import trace
Importing: C:/Users/Public/Documents/SD Tracing//logs/Audacity.exe/2010-04-02__01-34-23/profile_5888.sdt
Trace Name: Audacity.exe - 2010-04-02__01-34-23
Thread
Function Executions
0
21.542
Thread 0
Short name
Count
Max frequency
File
within 224 44 trackpanel.cpp (c:/projects/audacity/aud
DetermineToolToUse 260 44 trackpanel.cpp (c:/projects/audacity/aud
Dark 252 43 acolor.cpp (c:/projects/audacity/audacity
Bevel 252 43 acolor.cpp (c:/projects/audacity/audacity
ValueToControls 234 42 timetextctrl.cpp (c:/projects/audacity/aud
ControlsToValue 234 42 timetextctrl.cpp (c:/projects/audacity/aud
UpdatedEvent 120 28 track.cpp (c:/projects/audacity/audacity-
Init 120 28 track.cpp (c:/projects/audacity/audacity-
TP_DisplayStatusMessage 388 22 project.cpp (c:/projects/audacity/audacity
SetCursorAndTipWhenSelectTool 302 22 trackpanel.cpp (c:/projects/audacity/aud
SelectionHandleDrag 382 22 trackpanel.cpp (c:/projects/audacity/aud
8752 of 21542 function executions remain in this thread after excluding functions.
Exclude functions if their frequency exceeds 40 executions per million ticks
8.752 of 21.542 function executions remain in all selected threads after excluding functions.
cancel
import trace

5.3 Manage and Delete Traces

In the menu of SD Dev Edition you will find the Manage Traces Dialog.

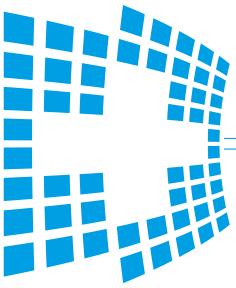
© 2010 Software Diagnostics



With it, you can remove traces from your SD project that belongs to your current Visual Studio solution.

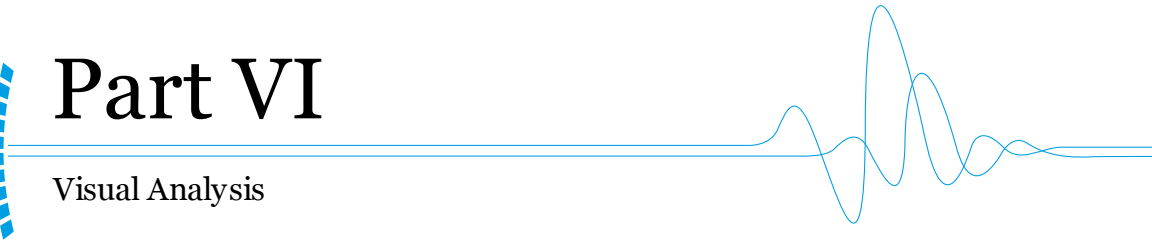
Furthermore, you can manually add traces. Traces are stored in %SD_TRACING%\logs\app.exe\<timestamp>. Choose a *.sdt to import.

© 2010 Software Diagnostics



Part VI

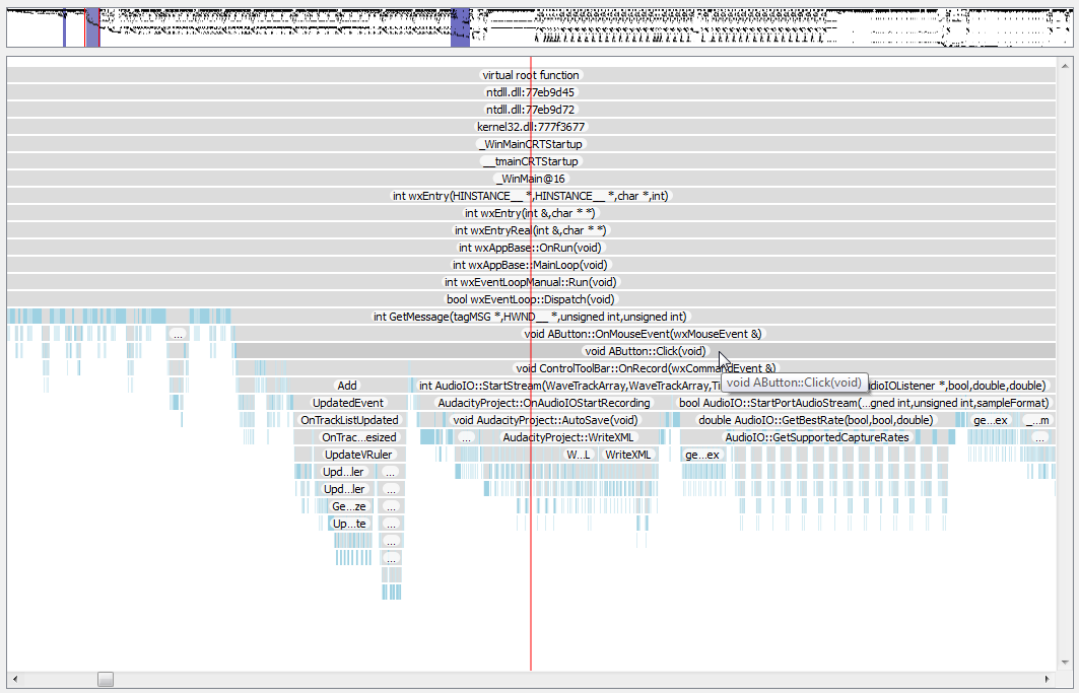
Visual Analysis



6 Visual Analysis

Imported traces are visualized by a set of synchronized views. The main views are the CallStackView, the CallGraphView, and the SingleCallView. Furthermore, the SearchView is helpful to quickly find entry points in the trace.

6.1 CallStackView



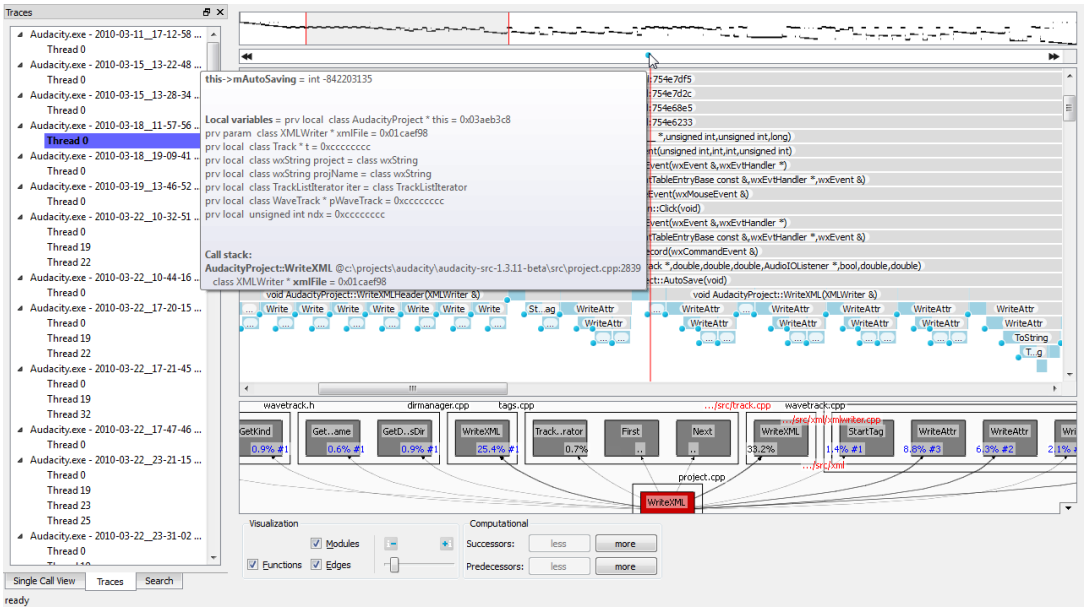
This view shows the trace in its temporal order. In the upper, overview part, the complete trace is depicted. You can identify the different phases in the execution.

The lower part displays a time range in detail. For orientation, this time range is marked in the overview.

Time is displayed horizontally from left to right. Bars are function calls. They are stacked which represents the call stack. In contrast to usual debuggers, this view permits the call stack to be analyzed not for a single point in time but for a full time range.

Using the mouse wheel, you can seamlessly adjust the time range shown.

© 2010 Software Diagnostics



If Probes are recorded they are shown in the CallStackView, too.

DataProbes and ParamProbes are attached to the calls as blue dots. MemAccessProbes appear on top; with the arrow symbol you can navigate to the points in the execution where the MemAccessProbe was hit.

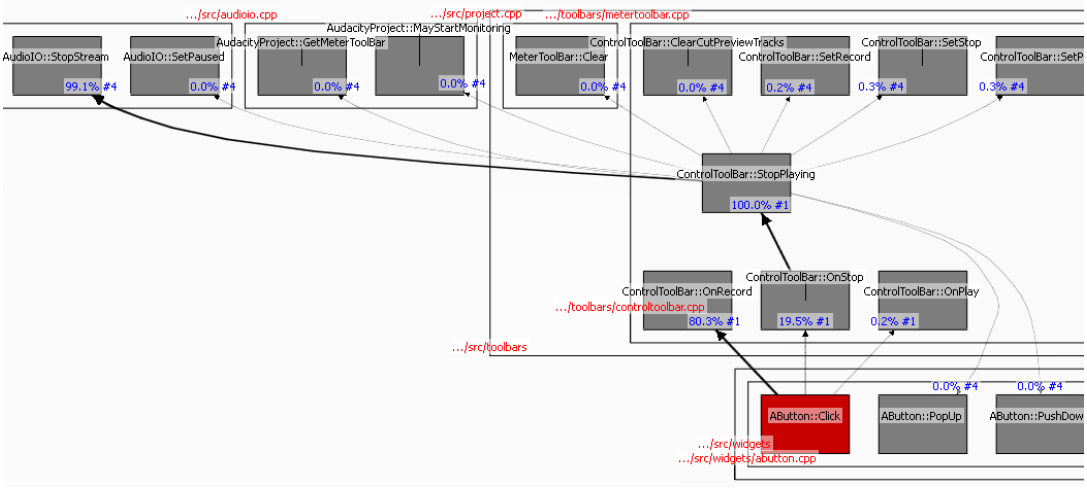
This view highlights all occurrences of a function that the mouse is pointing to.

Right clicking on a function offers you a context menu with options to navigate in the CallStackView, to create a call graph in the CallGraphView, to show the call in the SingleCallView, or to jump to Visual Studio's code view.

Execution time is not shown linearly. That is, you should use a different view if you are interested in performance.

© 2010 Software Diagnostics

6.2 CallGraphView



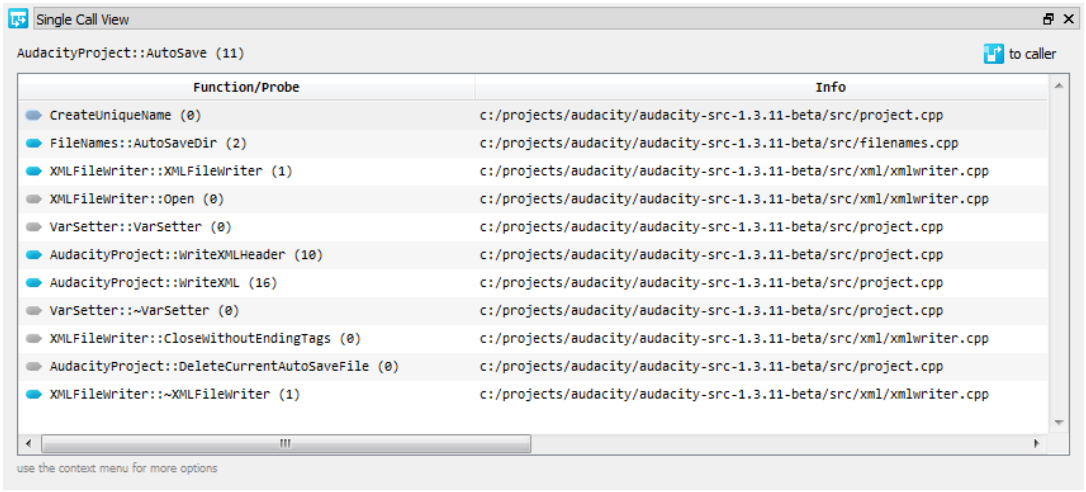
This view shows call relations between functions. Functions are embedded into their module structure, i.e., files and directories. Edges are annotated with information on the relative call costs and the call count.

Optionally, the call graph can be restricted to a specific time range. For this, use the context menu in the CallStackView. To remove a time constraint, click on the button that appears in the upper right corner of the CallGraphView.

Double-clicking on a function extends the graph by all called functions.

Double-clicking with the shift-key pressed, extends the graph by the functions that call the function that is clicked.

6.3 SingleCallView



This view shows all calls for a call "foo()" that are triggered by it, in chronological order.

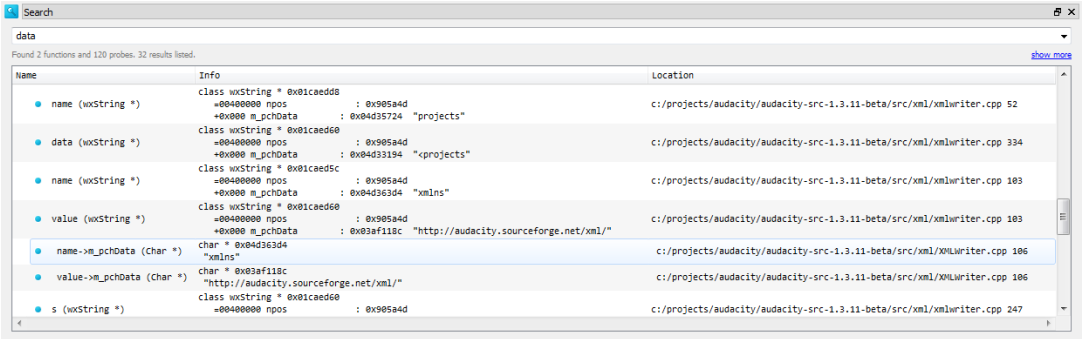
© 2010 Software Diagnostics

Hence, this is a condensed visualization of the code with all loop structures enrolled by their sequential execution order.

Cursor, enter, and backspace keys permit you to navigate to subcall or parent calls, respectively.

Right clicking on a function offers you a context menu with options to navigate in the CallStackView, to create a call graph in the CallGraphView, to show the call in the CallStackView, or to jump to Visual Studio's code view.

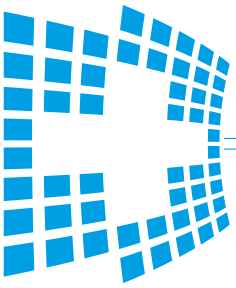
6.4 SearchView



This view enables the quick navigation within the execution to points of interest. All data contained in a trace is searched including function names, file names, and probes.

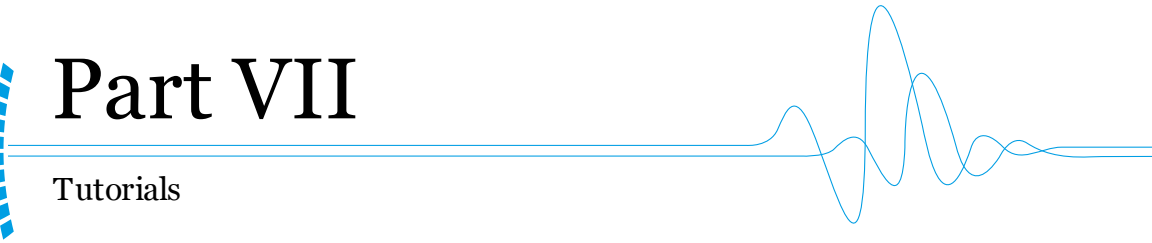
Right clicking on a function or probes permits you to focus other views on the selected function or probe.

© 2010 Software Diagnostics



Part VII

Tutorials



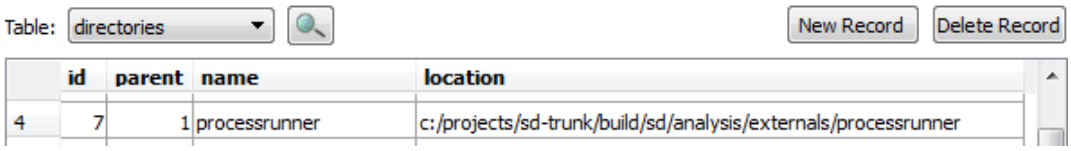
7 Tutorials

This chapter shows several case studies in the form of short tutorials to illustrate how SD DevEdition can be applied to your development process.

7.1 Bug: Erroneous Database Entry

Bug Description

Software Diagnostics Developer Edition is applied during its own development. One of our developers noticed that the hierarchical structuring of directories in the call graph view was wrong. Hence, he checked the database entry for the respective directory "processrunner" and saw that the entry "1" in the parent row was not correct (see screenshot).



Question and Hypothesis

Where in the code, when and why is this database entry set? During the import of a trace (`import()`), there is a method `importFunctions()` that indirectly also imports directory information. The developer's first guess was that the bug is located in this method. The table is filled with several hundred entries. Hence, the database is accessed frequently when the table is populated.

Approach with SD DevEdition

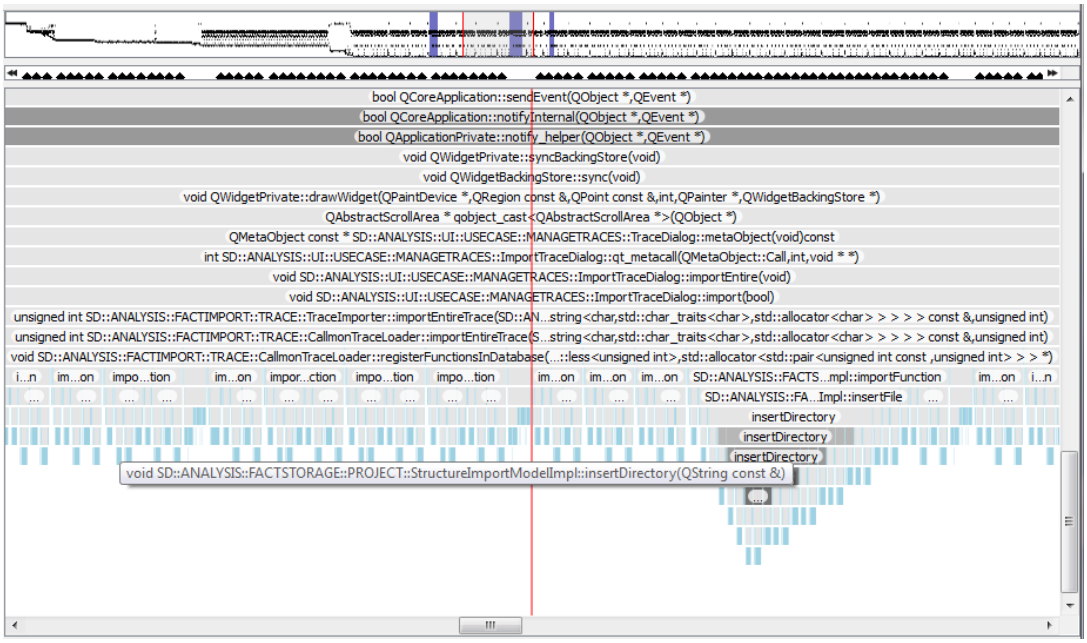
1. Set breakpoint in method `import()` (Microsoft Visual Studio).
2. Open the FunctionSelector Dialog and ensure that the suspicious functions are activated for tracing.
3. Add a DataProbe to record the SQL commands that are passed to the database and open the class that encapsulates the database communication in Microsoft Visual Studio:

```
void SQLiteImpl::execQuery(QSqlQuery& query, const QString& queryString)
{
    query.exec(queryString);
    lastInsertId_ = query.lastInsertId().toULongLong();
}
```

4. On the code line `query.exec()`, right click to add a DataProbe.
5. In the dialog, choose to observe the variable `queryString`.

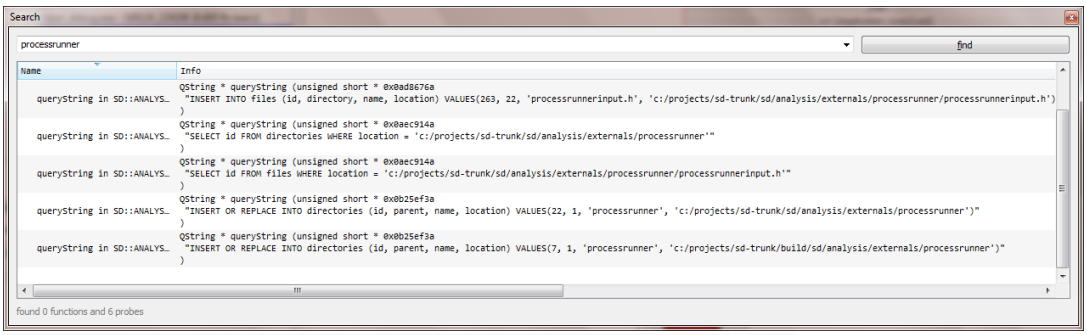
© 2010 Software Diagnostics

6. Start the application in the usual debugging mode of Microsoft Visual Studio [Menu->Debug->Start Debugging].
7. Interact with the application such that the import trace feature is activated.
8. If the breakpoint is hit, the application stops in method `import()`.
9. Perform a TraceOver. Thereby, move the stop breakpoint to the end of the `import()` method.
10. Trace without ParamProbes
11. Analyze the resulting trace (~40.000 calls and 1000 DataProbes were recorded):



Analysis results

- Some of the importFunction() calls indirectly trigger insertDirectory() as hypothesized
- Search for "processrunner" to find the DataProbe records that are related to the wrong database entry: this gives us 6 DataProbe records - the last entry shows how the erroneous database entry is inserted.

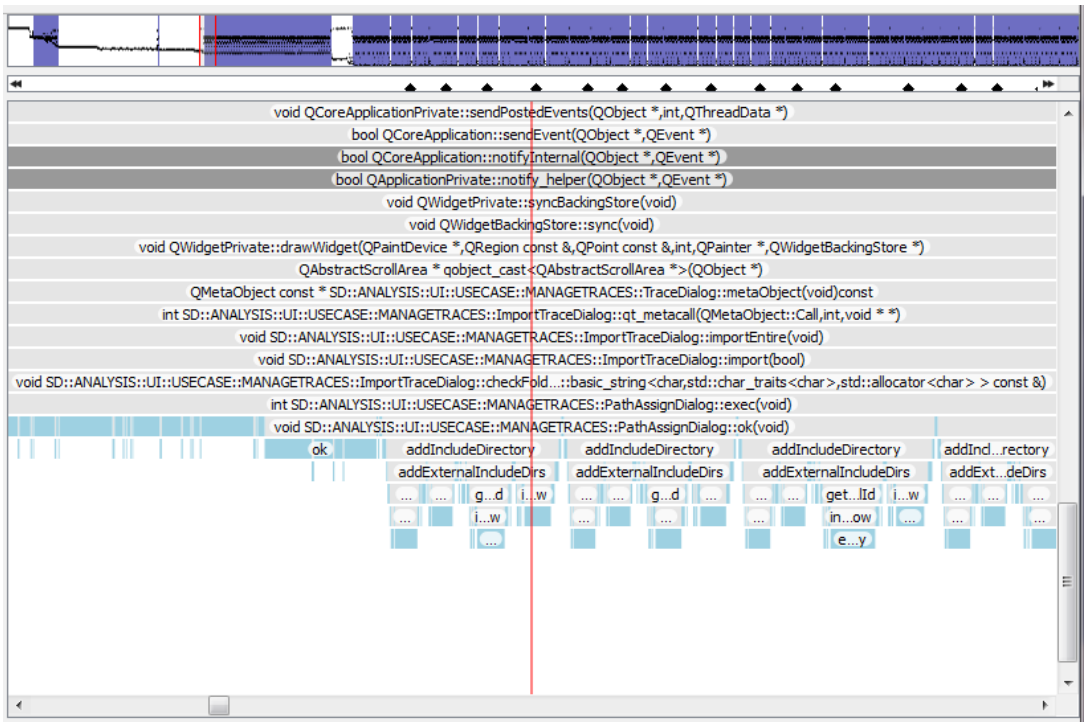



```
QString * queryString (unsigned short * 0x0b25ef3a
 "INSERT OR REPLACE INTO directories (id, parent, name, location)
)
```

© 2010 Software Diagnostics

```
VALUES(7, 1, 'processrunner', 'c:/projects/sd-trunk/build/sd/analysis/externals/processrunner')"
```

- Right click on the DataProbe record and choose “Jump to time”.
- The CallStackView shows an unexpected and surprising fact: the database entry is not inserted as part of importFunction() as hypothesized but much earlier in the execution. The entry comes in when a “PathAssignDialog” checks “include directories” - obviously the bug is located here.



Bug Solution

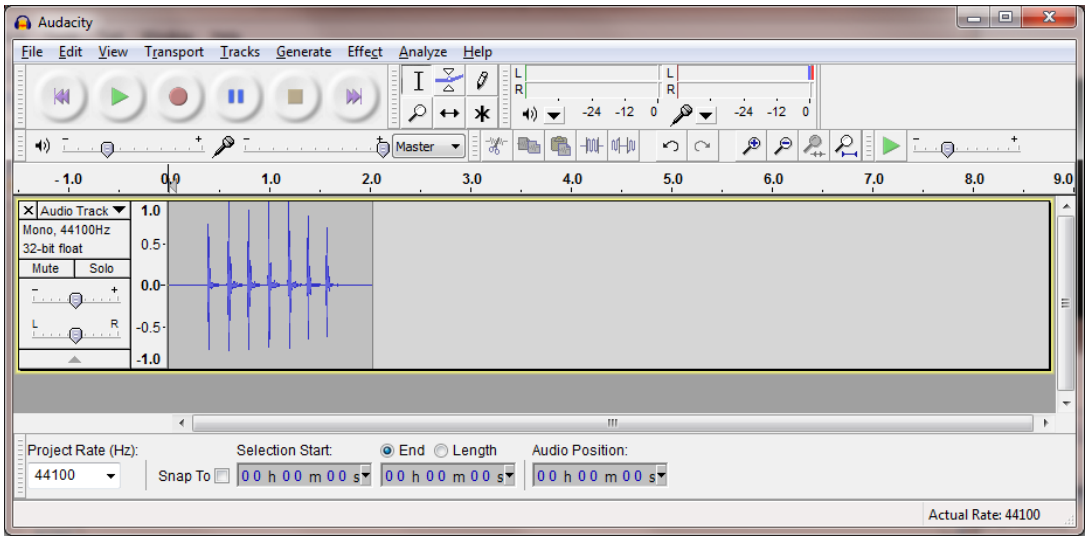
After that discovery, the developer set a breakpoint into the `ok()` method of the PathAssignDialog and continued usual debugging with Microsoft Visual Studio. He quickly identified the erroneous logic and fixed the bug.

7.2 Bug: XML Content Written in Wrong Order

The open-source audio tool Audacity comprises approximately 200.000 lines-of-code. When recording wave data, it triggers an autosave() function that automatically stores the current project settings. The settings are written into an XML file. However, the XML content was written in wrong order. SD Dev Edition was applied to analyze the autosave() function.

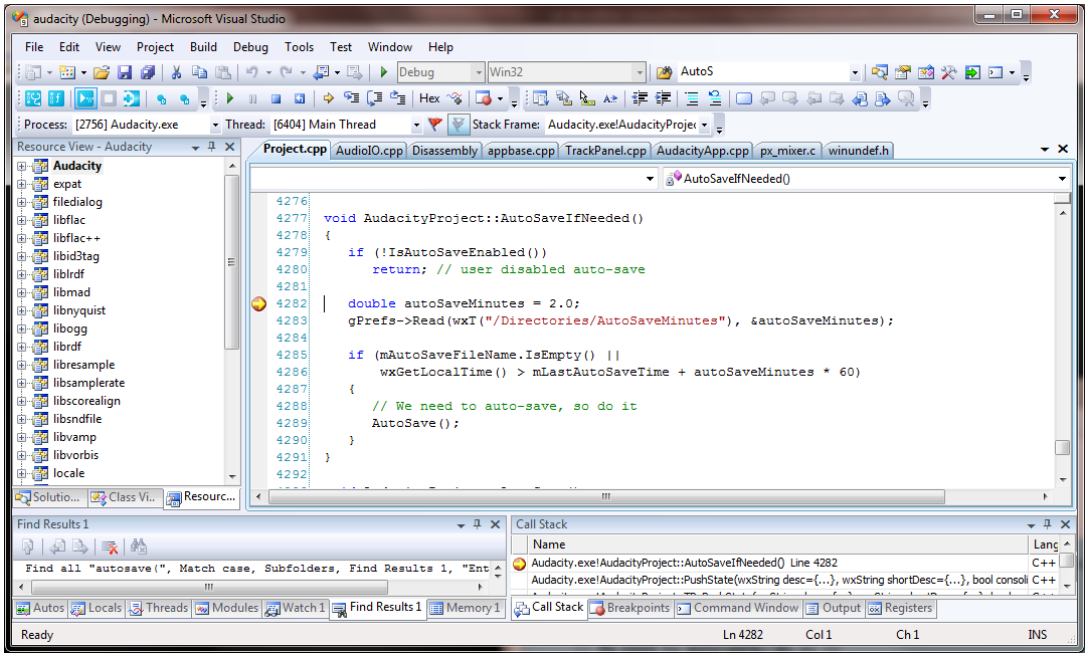
© 2010 Software Diagnostics

The trace data for this tutorial is part of the installer of SD Dev Edition. You'll find it in the help menu.



Tracing

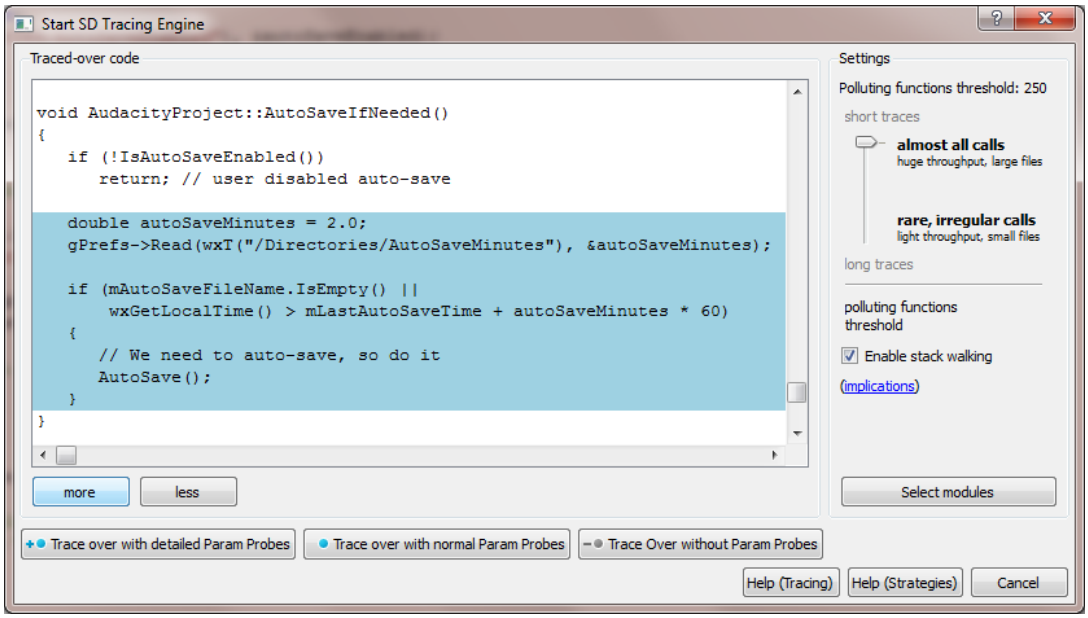
Audacity is started within the Microsoft Visual Studio debugger (F5). As we know, where the autosave() function is triggered, we set a breakpoint on the respective code line.



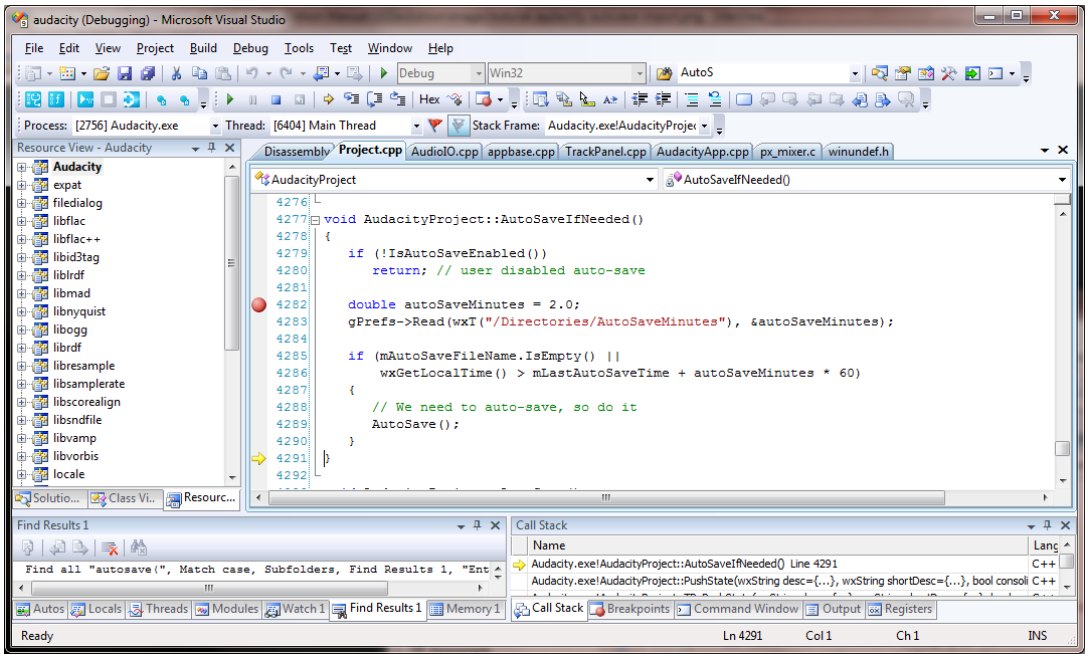
When the execution hits the breakpoint, we press the Tracing icon in the Microsoft Visual Studio SDAddin Toolbar. In the TraceOver dialog, we define the tracing endpoint (line 4291).

© 2010 Software Diagnostics

As we know that autosave() triggers less than 5.000 calls, we press "TraceOver with Detailed Param Probes".



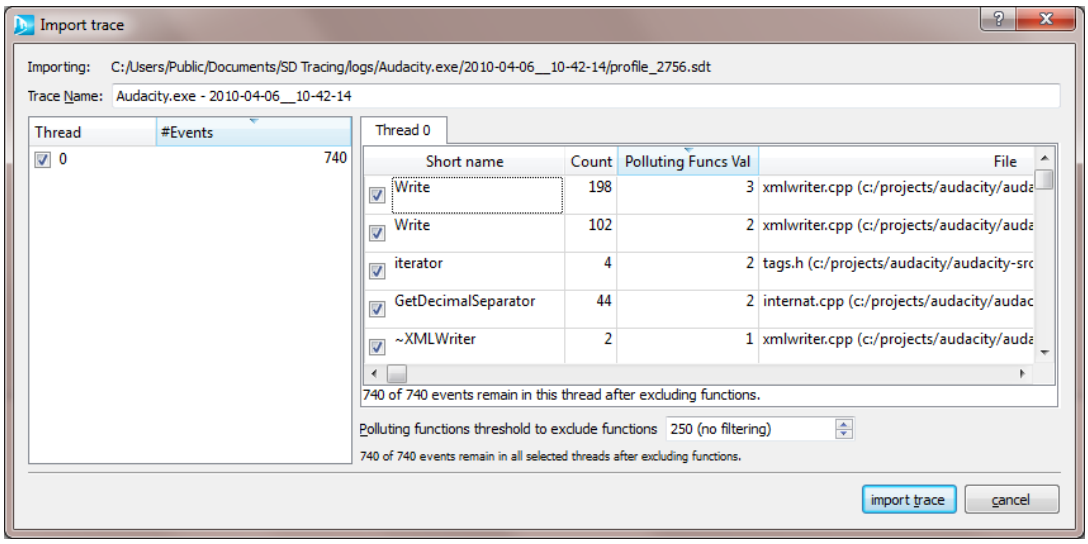
After TraceOver execution is stopped at the code line that was defined as tracing endpoint. The Microsoft Visual Studio debugger is attached to the process and ready for further commands.



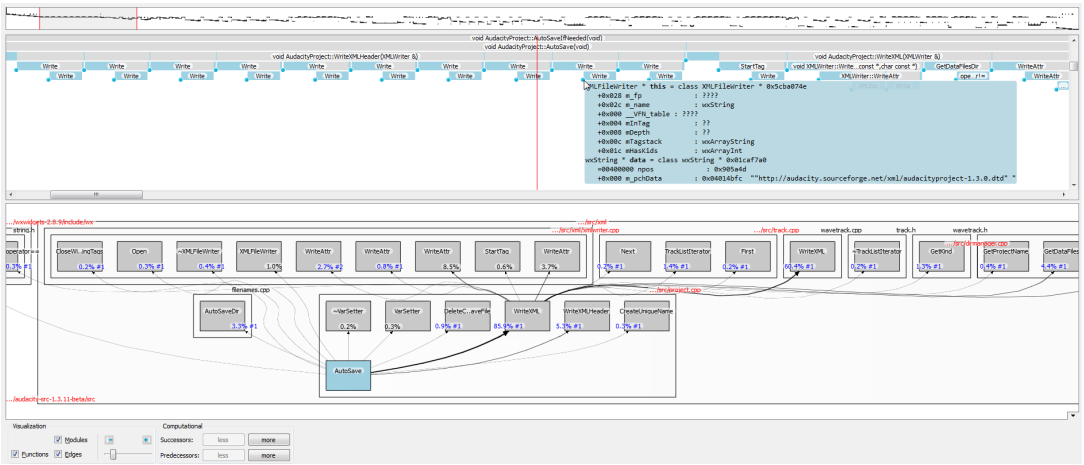
Visual Analysis

© 2010 Software Diagnostics

In the trace import dialog, we do not exclude functions and import the complete trace.



Next, we analyze in the CallStackView and the CallGraphView how the XML file is written. By pointing on the blue ParamProbes circles, we understand the order in which the strings are written into the XML file. As we are in particular interested to understand the context of writing the "http://audacity.sourceforge.net" string, we search for it in the SearchView, find the ParamProbe that contains the string, and jump to the point in time where the ParamProbe was recorded via the context menu on the search result.



© 2010 Software Diagnostics

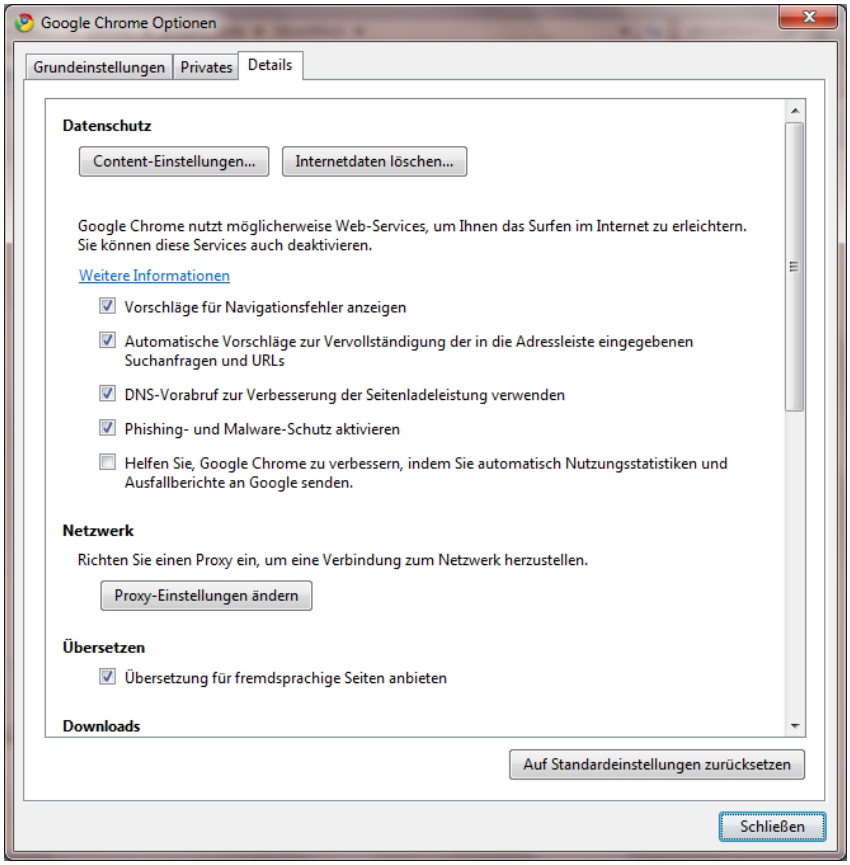
7.3 Feature Location: Privacy Data in Google Chrome

The source code of the open-source web browser Google Chrome comprises approximately 4 million lines-of-code. We were interested how Chrome collects privacy data on the user's usage behavior. Hence, this tutorial reflects a typical "feature location" task: Find feature-relevant code in weakly-known code.

The trace data for this tutorial is part of the installer of SD Dev Edition. You'll find it in the help menu.

Traced Feature

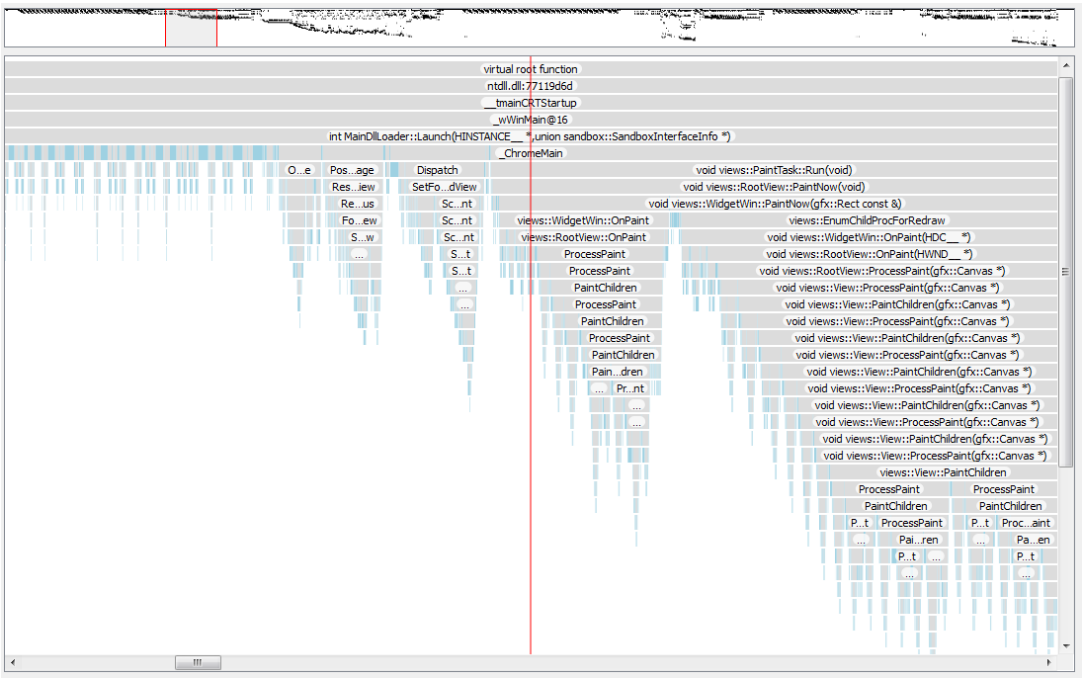
Trace Chrome while the user enables the privacy data feature in the browser's options dialog.


Google Chrome Optionen
Grundeinstellungen
Privates
Details
Datenschutz
Content-Einstellungen...
Internetdaten löschen...
Google Chrome nutzt möglicherweise Web-Services, um Ihnen das Surfen im Internet zu erleichtern. Sie können diese Services auch deaktivieren.
Weitere Informationen
Vorschläge für Navigationsfehler anzeigen
Automatische Vorschläge zur Vervollständigung der in die Adressleiste eingegebenen Suchanfragen und URLs
DNS-Vorabruf zur Verbesserung der Seitenladeleistung verwenden
Phishing- und Malware-Schutz aktivieren
Helfen Sie, Google Chrome zu verbessern, indem Sie automatisch Nutzungsstatistiken und Ausfallberichte an Google senden.
Netzwerk
Richten Sie einen Proxy ein, um eine Verbindung zum Netzwerk herzustellen.
Proxy-Einstellungen ändern
Übersetzen
Übersetzung für fremdsprachige Seiten anbieten
Downloads
Auf Standardeinstellungen zurücksetzen
Schließen

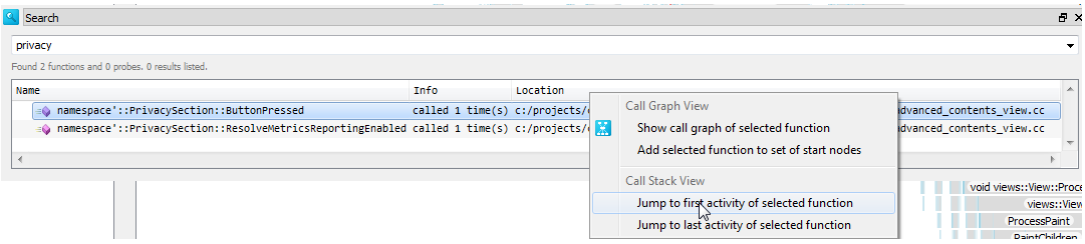
© 2010 Software Diagnostics

Visual Analysis

The trace reveals that there are different execution phases. Below you see the internal workings when the options dialog is painted.

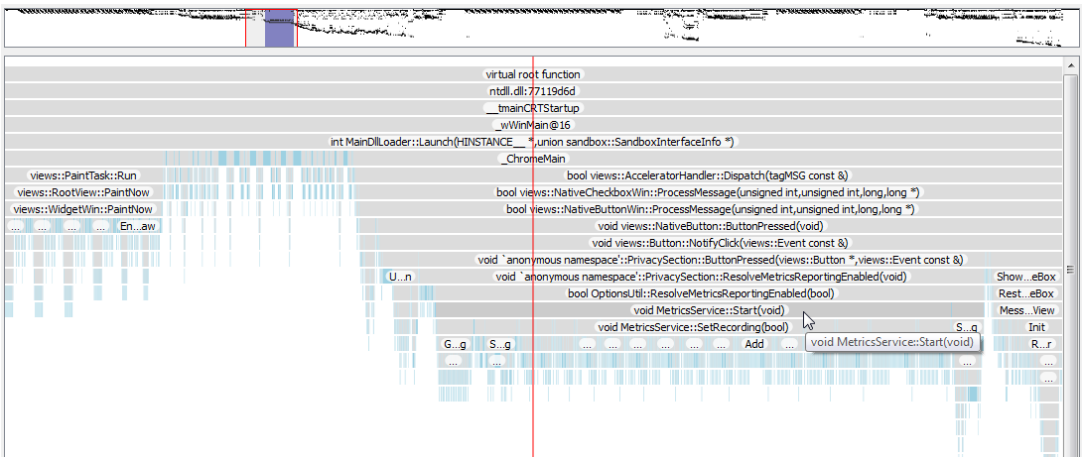


In the search view, we type "privacy" to find any function that is related to privacy data. We obtain two results. Via the context menu, we jump to the point in time where ButtonPressed() is called.

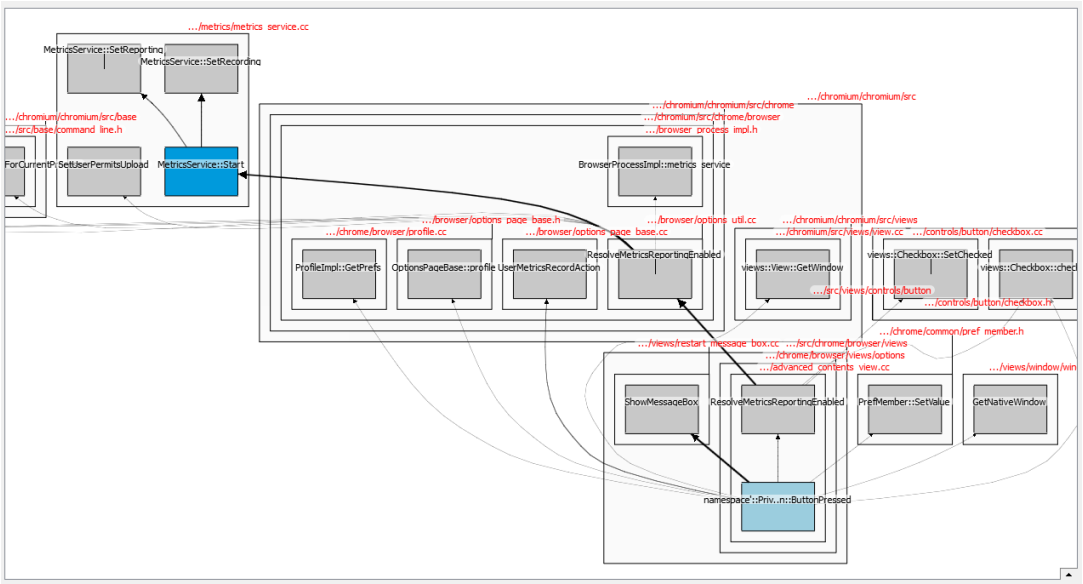


We quickly identify that the class MetricsService is responsible for collecting usage behavior. This service is started when the user enables the privacy data feature.

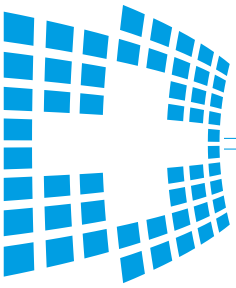
© 2010 Software Diagnostics



The CallGraphView helps to understand how the control flow is passed from the "button click" to the start of the MetricsService.

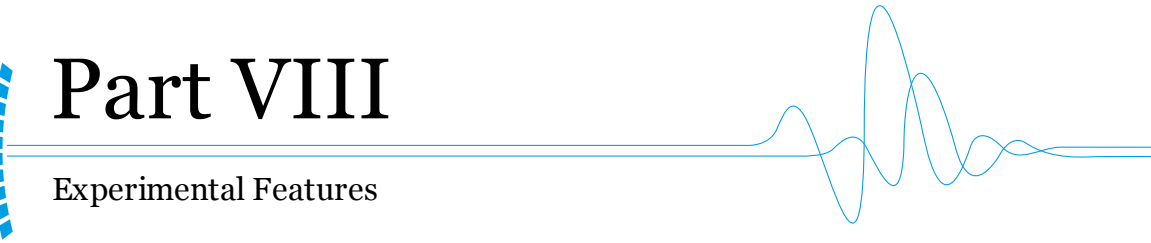


© 2010 Software Diagnostics



Part VIII

Experimental Features



8 Experimental Features

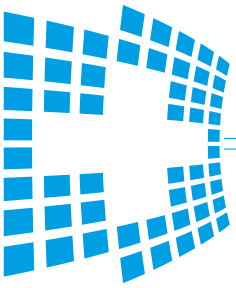
Some of the features of SD Dev Edition are experimental.

- Tracing optimized code:
  Tracing your application in the Release configuration might crash your application in specific circumstances. To avoid this, trace your application in the Debug configuration.

- MemAccessProbes:
  The size of the memory location that is observed by a MemAccessProbe cannot be resolved automatically in specific circumstances. Setting a wrong size might cause SD Dev Edition to ignore the MemAccessProbe.

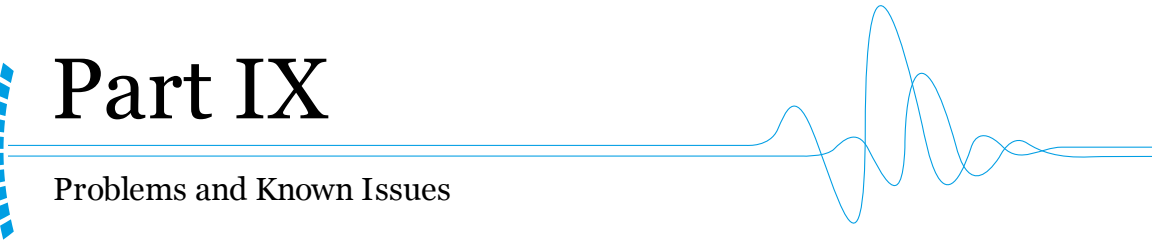
- DataProbes on Global Variables:
  Obtaining the list of global variables in the DataProbe dialog might take a long time.

© 2010 Software Diagnostics



Part IX

Problems and Known Issues



9 Problems and Known Issues

Please send us an email if you have a feature request or any other idea to improve SD Dev Edition: service@softwarediagnostics.com

Possible Interferences with Other Development Tools

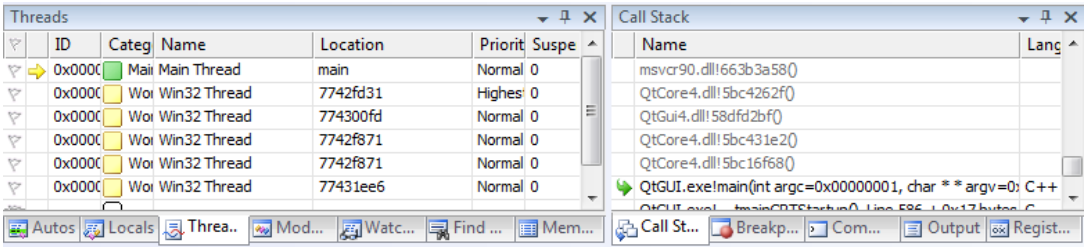
- If you are using IncrediBuild (Xoreax Ltd.), you might need to disable its "distributed linking" feature.

Tracing Issues

- Tracing stops immediately after starting . There might be the error message that your application crashed.
  Open folder %SD_TRACING%\logs\<your-app>. Delete all *.cmd files.

- Your app hangs and cannot be killed:
  Kill all cdb processes.

- After TraceOver, Microsoft Visual Studio does not display the call stack and the threads correctly:
  Double-click on another thread and than on the original back. The call stack and thread window is now as usual.

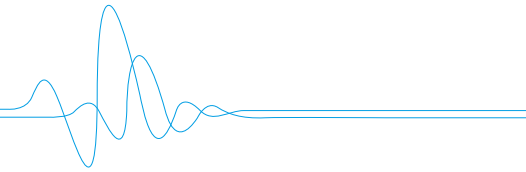


  Another solution is to detach Microsoft Visual Studio from the process and attach it again.

- The SolutionPreparation Dialog says that your Visual Studio Project Files are corrupt:
  There are VS projects in your VS solution that refer to VS configurations, e.g., Debug|x64, that do not exist. Please open <%MS_VS%>'s configuration manager dialog (in the build menu) dialog and check that all projects refer to valid configurations.

Probes Issues

© 2010 Software Diagnostics



- DataProbes or MemAccessProbes cannot be set:
  You might try to set a Probe on a code line that is not part of the executable that is currently set as StartUp Project your Microsoft Visual Studio solution. Ensure that the correct StartUp Project is chosen.
  Additionally, DataProbes and MemAccessProbes are not supported for dynamically loaded DLLs at runtime, e.g., with the LoadLibrary() Windows function.

- DataProbes or MemAccessProbes cannot be deleted:
  Clicking on a Probe to delete it just toggles a usual breakpoint. Close the respective file and open it again. After this, deletion should work as expected.

Visual Analysis Issues

- Mysterious Call Relations:
  If C++ exceptions occurred during tracing, the call stack is not guaranteed to be recorded in the correct order. Hence, you might see mysterious call relations in the trace in these situations.

- The CallStackView or the CallGraphView is gone:
  You can move the line between the two views by dragging it with the mouse. Moving the line to the top fully hides the CallStackView. Moving it to the bottom hides the CallGraphView. To make it visible again, grab the line at the top or bottom, respectively, and move it to the center again. On some Microsoft Windows platforms, the line is invisible (due to Windows standard GUI style).

- The labels in the CallGraphView are distorted:
  This is probably a problem of your graphics card driver. Please download the latest driver from the homepage of your graphics card vendor and install it. Another solution is to disable 3D graphics support in general or only for the executable "sd_tracer.exe", which is placed in the %SD_HOME% folder. The option for disabling 3D graphics support is typically at the same dialog where you can change the screen resolution.

- Call markers, DataProbes, and MemAccessProbes are not shown in the Microsoft Visual Studio code view:
  Close the respective source code file and open it again. You may open it with Microsoft Visual Studio or via the context menu on the respective function or call in SD Dev Edition.

- The context menu entry for switching to Microsoft Visual Studio is missing in the CallStackView and the CallGraphView:
  You have started SD Dev Edition as stand-alone tool. Close SD Dev Edition, start Microsoft

© 2010 Software Diagnostics

Visual Studio, and start SD Dev Edition again by clicking the respective icon in the toolbar.

© 2010 Software Diagnostics

software diagnostics

© 2010 Software Diagnostics